Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Sergi Grau Dalmau

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: Desarrollo de aplicación web para la gestión del consumo en la comunidad de vecinos Carme Laforet

Director/a: Josep Argelich

Presentació

Mes: Juny

Any: 2018

Índice General

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia todo el apoyo recibido por su parte durante todos los años del grado, ya que han sido diferentes etapas de subidas y bajadas emocionales, las cuales he sabido gestionar correctamente gracias a ellos.

En segundo lugar, solo puedo que agradecer a cada uno de los profesores que he ido teniendo durante toda esta etapa, ya que he aprendido mucho de ellos y con ellos, gracias a sus ganas de enseñar y vernos mejorar.

Por último, dar las gracias a todos los compañeros que ido conociendo durante todo este transcurso ya que esta etapa se me ha hecho más amena gracias a ellos.

Resumen

El siguiente proyecto de final de grado, consiste en el desarrollo de una aplicación web *Responsive* [1] con la que se pretende sustituir la función que por el momento ocupa el administrador de fincas, de la comunidad de vecinos *Carme Laforet*, la cual consiste en llevar todos los cálculos y la administración, de los gastos de agua, luz y calefacción, de los 4 bloques que dispone dicha comunidad.

Permitiendo de esta forma tanto a la persona encargada de dichas tareas como a los propios vecinos, una mejor consulta, organización y almacenamiento de todos los datos mensuales e históricos de los cuales, gracias a este proyecto, dispondrán los vecinos en todo momento y en cualquier lugar, siempre y cuando dispongan de conexión a internet.

[1] Consiste en una técnica de diseño web que busca la correcta visualización de una página en distintos sitios.

Capítulo 1: Introducción

En este capítulo se expondrán los objetivos que se quieren lograr con dicho proyecto, junto con la evolución de la planificación de las tareas estipuladas para la creación de la aplicación. Se detallará el tiempo que finalmente se ha empleado para realizarlo. Por último, también se presentará un presupuesto final orientativo del coste del desarrollo íntegro de la aplicación.

Objetivos

El principal objetivo del proyecto es diseñar una aplicación web, la cual se encargue de sustituir y mejorar, incluso, las tareas que actualmente está realizando un administrador de fincas, las cuales consisten en:

- Recopilar toda la información (*gastos luz, gastos agua, gastos calefacción, perdidas…*).
- Realizar todos los cálculos necesarios para obtener las facturas individuales para cada vecino.
- Poner al alcance de cualquier vecino, toda la información mensual de la comunidad.
- Almacenar la información mensual, en un motor de base de datos, el cual siempre tendrá la información disponible para ser consultada por cualquier vecino, a través del apartado de históricos de la aplicación.
- La versión 2.0 de la aplicación permitirá realizar el pago de la mensualidad a cada vecino, a través de una cuenta *PayPal* vinculada a la cuenta de la comunidad.
- Diseñar una aplicación *multidispositivo* [2].

Motivación

Para que engañarnos, como cualquier alumno o la gran mayoría, cuando llegó el momento de escoger el tema sobre el que desarrollar el *Trabajo de Final de Grado* no tenía nada claro en que centrarme, lo único que tenía en mente era que quería realizar una aplicación informática que me permitiera demostrarme a mi mismo que realmente todo lo aprendido durante el grado me puede servir para llevar a cabo ideas que facilitan el día a día, ya sea de empresas, comunidades o colectivos de personas…

Entonces encontré una oferta muy llamativa basada en el desarrollo de un aplicativo web *multidispositivo Responsive*, tema el cual ya habíamos tratado durante mis años de formación en el grado pero a bajo nivel, entonces me propuse desarrollar dicha idea intentado llevarme a los límites del *framework* [3] utilizado para que a la vez de facilitar, en este caso a la comunidad de vecinos su mensualidad, también yo poder aprender el uso completo de este.

---

[2] Aplicación accesible desde cualquier dispositivo.

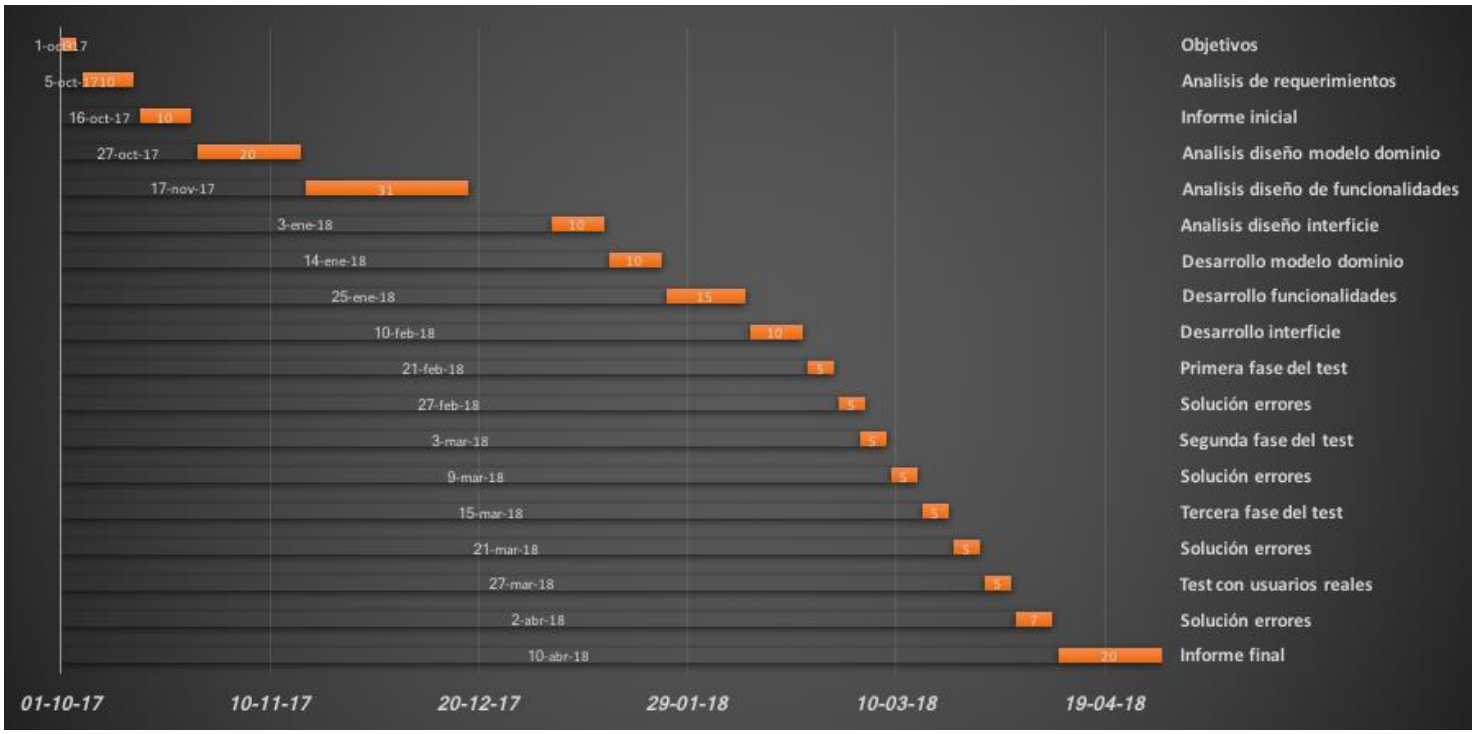
[3] Esquema o patrón para el desarrollo y/o la implementación de una aplicación.

Temporalidad

Para estimar el tiempo necesario en el desarrollo del proyecto, se han dividido las tareas en 4 partes fundamentales:

- ***Análisis Requerimientos***: Dónde se ha realizado un análisis exhaustivo de las necesidades y requerimientos que se necesitan cumplir con el desarrollo de dicha aplicación.
- ***Análisis Diseño***: Una vez realizado el análisis de cómo y que se necesita, se ha diseñado la forma de obtener dichos propósitos junto con todo el modelo de datos que se encargará de dar coherencia a todo el proyecto.
- ***Desarrollo***: En esta tarea se ha realizado el desarrollo de todos los casos estudiados en las anteriores dos tareas.
- ***Testeo***: Por último se han llevado a cabo múltiples testeos de la aplicación, intentando simular todos los casos posibles que se pueden dar en su uso cotidiano.

Cada una de estas tareas, posteriormente también ha sido subdividida, tal y como se muestra en el siguiente *Diagrama de Gantt*:



Presupuesto

Para obtener una estimación del coste del desarrollo del proyecto, a parte del tiempo e ideas empleadas, también se han tenido en cuenta las herramientas utilizadas.

Cabe recalcar que siempre se han utilizado herramientas de coste libre, aunque en la última etapa del proyecto se ha utilizado *PyCharm* [4] con licencia educativa debido que se ha encontrado necesario el uso de la ejecución de la aplicación en modo *Debug* [5].
Seguidamente se muestra un cuadro aproximativo del presupuesto desglosado:

Artículo	Precio (€)	Cantidad
Microsoft Excel	0	1
ATOM	0	1
PyCharm	0	1
PC	1000	1
Cacoo	0	1
Pencil	0	1
Servidor VPS	20/Mes	1
Desarrollador	12	700
Total	**9400**	

En el anterior cuadro, no se han tenido en cuenta los costes de mantenimiento del aplicativo, debido a que todavía queda por acordar con la comunidad de vecinos las condiciones de compra de dicha aplicación.

A parte, se tiene en mente sacar una segunda versión del aplicativo con un conjunto de mejoras, las cuales aumentarán el coste total, pero que por el momento prefiero dejar al margen.

Finalmente, como se puede apreciar, en el total no hay contabilizado el coste del servidor *VPS* [6], debido a que se trata de un gasto mensual que ha asumido desde primer momento la comunidad de vecinos.

---

[4] Entorno de programación Python propiedad de IntelliJ

[5] Herramienta para depurar la ejecución de un programa.

[6] Servidor Virtual Privado.

Capítulo 2: Tecnologías empleadas

En este capítulo se enumerarán todas las tecnologías empleadas para el correcto desarrollo de la aplicación, junto también con una pequeña explicación de el porque se ha escogido cada tecnología y el uso del software, terminando con una breve explicación sobre los estudios previos al proyecto, realizados.

La aplicación web desarrollada, consta de dos partes claramente diferenciadas, las cuales son el *Front-end* [7] y el *Back-end* [8].

Back-end

En este apartado se enumerarán los aspectos más importantes que forman parte del Back-end de la aplicación.

- *"Django Framework"*

[1] **Django** es un *framework* de desarrollo web de código abierto, escrito en *Python*, que respeta el patrón de diseño conocido como *Modelo – Vista – Controlador* [9].

Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la *World Company de Lawrence*, Kansas, y fue liberada al público bajo una *licencia BSD* [10] en julio de 2005. El *framework* fue nombrado en alusión al guitarrista de jazz gitano *Django Reinardt*.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio "*No te repitas*" (*DRY, del inglés Don't Repeat Yourself*).

Python es usado en todas las partes del *framework*, incluso en configuraciones, archivos, y en los modelos de datos.

Elección

Uno de los motivos por el que he escogido el *framework Django* es que sigue el patrón de diseño *Modelo-Vista-Controlador*, ya que permite una alta reutilización de código, una mejor conectividad y extensibilidad de componentes, permitiendo de esta forma un desarrollo más rápido.

---

[7] Consiste en la parte del software que interactúa con los usuarios.

[8] Consiste en la parte del software que procesa la entrada de datos provenientes del Front-end.

[9] Patrón de la arquitectura del software.

[10] Licencia de software otorgada por Berkeley Software Distribution.

Pero el principal motivo que me ha hecho decidir en la elección de *framework, ha sido que Django* disponga de un modelo de datos ORM [11] (*Object Relational Mapping)*, el cual permite relacionar directamente las clases creadas en el código Python con la base de datos utilizada, facilitando enormemente de esta forma el manejo y el acceso de la base de datos.

- *"PostgreSQL"*

[2] ***PostgreSQL*** es un sistema de gestión de base de datos relacional orientado a objetos y libre, publicado bajo la *licencia PostgreSQL*, similar a la *BSD* o la *MIT* [12].

Como muchos otros proyectos de código abierto, el desarrollo de *PostgreSQL* no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre o apoyados por organizaciones comerciales. Dicha comunidad es denominada el *PDGD* (*PostgreSQL Global Development Group)*.

Elección

Se ha escogido este motor de base de datos, debido a que permite una alta concurrencia mediante el sistema denominado *MVCC (Acceso concurrente multidiversión)* que permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.

Otro motivo es que no genera ninguna inconsistencia con el S.O. *CentOs* utilizado y que junto con *MySql* es el motor de base de datos más recomendado por los expertos en un entorno de producción *Django*.

- *"Servidor Apache"*

[3] ***Apache*** es un servidor HTTP de código abierto, para plataformas UNIX,GNU/Linux,Windows, Mac y otras, que implementa el protocolo HTTP y la noción de sitio virtual.
El servidor Apache es desarrollado y mantenido por una comunidad de usuarios

[11] Modelo de programación que transforma las tablas de una base de datos en entidades.

[12] Licencia de software otorgada por el Instituto Tecnológico de Massachusetts.

bajo la supervisión de la *Apache software foundation* dentro del proyecto HTTP Server (httpd).

Apache es usado principalmente para enviar páginas web estáticas y dinámicas en la *World Wide Web* [13].

Muchas aplicaciones web están diseñadas asumiendo como ambiente de implantación a Apache, o que utilizarán características propias de este servidor web.

Elección

He escogido este servidor para lanzar mi aplicación a producción, debido a que dispone del módulo *WSGI* el cual permite una correcta vinculación del *framework Django* con *Apache*.
Otro punto a favor en la elección de *Apache* ha sido mi ligera familiarización con este software.

- *"Python"*

[4] **Python** es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la *Python Software Fundation*. Posee una licencia de código abierto, denominada *Python Software Foundation License*, que es compatible con la *Licencia Pública General de GNU* a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Elección

En este caso, sí que no ha habido elección ya que este es el lenguaje utilizado por el *framework Django*, aunque en el caso de haberla tenido, por su simplicidad a la hora de escribir el código, su fácil legibilidad y junto que es con uno de los lenguajes con los que estoy más familiarizado, hubiera escogido este mismo.

[13] Sistema de distribución de documentos hipertexto interconectados y accesible vía internet.

- *"Servidor VSFTPD"*

[5] ***VSFTPD***, es un servidor FTP para sistema de tipo UNIX, incluido Linux. Está licenciado bajo la *Licencia Pública General de GNU*. Es compatible con *IPv6* y *SSL*.

Dicho servidor es el predeterminado en las distribuciones *Ubuntu*, *CentOs*, *Fedora*, *NimbleX*, *Slackware* y *RHEL Linux*.

Elección

Tal y como se ha comentado en el apartado anterior, este servidor es el predeterminado para la distribución *CentOs*, por lo que este es el motivo que me ha hecho decidir en mi elección de FTP.

- *"Distribución CentOs 7"*

[6] ***CentOs*** es un sistema operativo de código abierto, basado en la distribución *Red Hat Enterprise Linux*, operándose de manera similar, y cuyo objetivo es ofrecer al usuario un software de "clase empresarial" gratuito. Se define como robusto, estable y fácil de instalar y utilizar. Desde la versión 5, cada lanzamiento recibe soporte durante diez años, por lo que la actual versión 7 recibirá actualizaciones de seguridad hasta el 30 de junio de 2024.

Elección

El motivo principal por el que he escogido esta distribución consiste en que en una de las asignaturas del grado, utilicemos dicho sistema operativo a pequeña escala y me quedé con ganas de indagar más en él.

Front-end

En este apartado se enumerarán los aspectos más importantes que forman parte del *Fron-end* de la aplicación.

- *"HTML 5"*

[7] **HTML**, que significa *Lenguaje de Marcado para Hipertextos* (*HyperText Markup Language*) es el elemento de construcción más básico de una página web y se usa para crear y representar visualmente una página web. Determina el contenido de la página web, pero no su funcionalidad.

Este lenguaje esta basado en un *Markup* [14] para anotar textos, imágenes y otros contenidos que se muestran en el navegador web.

En este caso se ha utilizado la versión 5 de este lenguaje, debido a que esta nueva versión proporciona grandes mejoras, unas de las cuales presentan dos conceptos diferentes:

- Se trata de una nueva versión de HTML, con nuevos elementos, atributos y comportamientos.
- Contiene un conjunto más amplio de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance. A este conjunto se le llama *HTML5 y amigos*, a menudo reducido a *HTML5* .

Elección

Se ha escogido este lenguaje para representar prácticamente todo el *Front-end* de la aplicación, debido a que se trata, en mi opinión, de la mejor opción que hay por el momento a parte de que es el que goza de mayor soporte.

- *"JavaScript"*

[8] **JavaScript** *(JS)* es un lenguaje de programación interpretado, dialecto del estándar *ECMAScript*. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz del usuario y páginas web dinámicas, aunque también existe una forma de JavaScript del lado del servidor (*server-side*).

[14] Lenguaje basado en marcas de hipertexto.

El código JavaScript es soportado por todos los navegadores que se encargan de interpretarlo.

Elección

Se ha escogido este lenguaje debido a que no tiene problemas de compatibilidad con ningún navegador y es el más utilizado en la creación de páginas web dinámicas.

- *"Bootstrap 4"*

[9] **Bootstrap** es un *framework* originalmente creado por *Twiter*, que nos permite crear interfaces web con *CSS* y *JavaScript*, cuya particularidad es la de adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. Es decir, el sitio web se adapta automáticamente al tamaño de un PC, una Tablet u otro dispositivo. Esta técnica de diseño y desarrollo se conoce como *Responsive Design* o diseño adaptativo.

A parte este framework contiene una enorme cantidad de clases que proporcionan tipografía, formularios, botones, cuadros, menús de navegación y una gran cantidad de elementos de diseño…

Elección

Se ha escogido esta herramienta, ya que uno de los objetivos de la aplicación era que esta fuera *multidispositivo* y según mi opinión y la de la gran mayoría del mundo de la informática, este *framework* te proporciona las facilidades y ayudas necesarias para lograr el objetivo.

- *"CSS"*

[10] ***Hojas de estilo en cascada*** (*CSS, Cascading StyleSheets*) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje marcado, como en este caso es HTML. Es muy usado para establecer el diseño visual de documentos web.

Junto con *HTML* y *JavaScript*, *CSS* es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y *GUIs* [15] para muchas aplicaciones móviles (como Firefox OS).
CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o *layouts* [16], los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características presentacionales, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .*CSS*, y reducir la complejidad y la repetición de código en la estructura del documento.

Elección

Se ha escogido CSS, porque es un estándar global en la creación de contenido web.

- *"JQuery"*

[11] ***JQuery*** es una biblioteca multiplataforma de *JavaScript*, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el *árbol DOM* [17], manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.
Es de software libre y de código abierto, posee un doble licenciamiento bajo la "*Licencia MIT*" y la "*Licencia Pública General de GNU*", permitiendo su uso en proyectos libres y privados. *JQuery*, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

---

15 Interfaz gráfica de usuario.

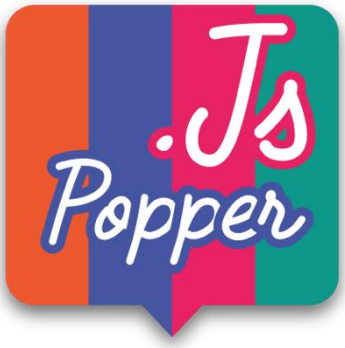
16 Esquema de distribución de los elementos dentro de un diseño.

17 Esquema estructurado HTML.

Elección

Se ha escogido esta biblioteca, debido a que esta fuertemente ligada con el *framework bootstrap*, ya que muchas de las facilidades que este proporciona se obtienen a través de *JQuery*.

- *"Popper JS"*



[12] ***Popper*** es una librería de JavaScript para añadir *tooltips* y *popovers* en elementos HTML. Ofrece un montón de opciones de personalización y es totalmente modular, con diferentes plugins para cada característica.

Elección

Se ha elegido esta librería para la creación de popovers, debido a que el tipo de popover a crear requería de su uso combinado con JQuery y Bootstrap.

- *"Font Awesome"*

[13] ***Font Awesome*** es un *framework* de iconos vectoriales y estilos css, que se utiliza para sustituir imágenes de iconos comunes por gráficos vectoriales convertidos en fuentes. Para ello utiliza una librería de más de 400 iconos transformados en fuentes.

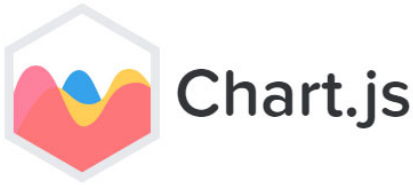
A parte de que nos permite reducir significativamente el uso de imágenes para iconos (lo cual se traduce en mejorar la velocidad de nuestra web).

Es compatible con la mayoría de dispositivos y navegadores, ya que únicamente requiere del soporte de "*@font-face*" por parte de los navegadores.
Al tratarse de iconos vectoriales nos permite escalar nuestros iconos sin problemas de resolución.
Font awesome es totalmente libre para su uso comercial.

Elección

Se ha elegido este *framework* para representar los iconos del aplicativo web, ya que dispone de una amplia librería de iconos que cumplen todas las necesidades requeridas, junto con su alta escalabilidad.

- *"Chart js"*



[14] ***Chart JS*** se trata de una de las librerías más populares para la creación de diferentes tipos de gráficos.
Esta librería te permite mezclar diferentes tipos de gráficos y trazar datos en escalas fecha tiempo, logarítmica o personalizada con facilidad. A parte también soporta animaciones que pueden ser aplicadas cuando se cambian los datos o se actualizan los colores.

Elección

Se ha escogido esta librería para la representación de los gráficos, ya que el material que proporciona va perfectamente ligado a las necesidades que requiere la aplicación en este ámbito.

Software

En este apartado se enumerará el software utilizado para el diseño y desarrollo del proyecto.

- *"ATOM"*

[15] ***ATOM*** es un editor de código fuente de código abierto para *macOS*, *Linux* y *Windows* con soporte para plugins en Node.js y control de versiones *Git* integrado, desarrollado por Github.
Es una aplicación de escritorio construida utilizando tecnologías web.
La mayoría de los paquetes tienen licencias software libre y están desarrollados y mantenidos por la comunidad de usuarios.

Elección

Se ha escogido este programa por su licencia libre y por su alta personalización en el uso de diferentes plugins que permiten configurarlo al gusto del programador.

- *"PyCharm"*

[16] **PyCharm** es un entorno de desarrollo integrado (IDE) utilizado en la programación de computadoras, específicamente para el lenguaje Python. Es desarrollado por la compañía checa *JetBrains*.
Proporciona análisis de código, un depurador gráfico, un probador de unidades integrado, integración con sistemas de control de versiones (VCS) y admite el desarrollo web con *Django*.

Es una multiplataforma, con versiones de *Windows*, *macOS* y *Linux*. La edición de la comunidad se publica bajo la *Licencia de Apache*.

Elección

Se ha escogido este programa para la fase de testeo, debido a que encontré necesario depurar varias fases de las funcionalidades de la aplicación, entonces aproveché que disponía de la licencia educativa para este programa y que previamente ya estaba familiarizado con él.

- *"Herramienta Cacoo"*

[17] **Cacoo** es una herramienta en línea que permite crear de manera colaborativa una variedad de organizadores gráficos tales como mapas mentales, *wireframes* [18], diagramas UML, y de redes, entre otros. Esta herramienta ofrece una licencia de uso gratuito y otro comercial.

Elección

Se ha escogido esta herramienta debido a que ya me encontraba familiarizado con ella, gracias a su uso durante el transcurso del grado.

---

[18] Boceto en el que se representa visualmente, de una manera muy sencilla y esquemática, la estructura de una página web.

- *"Herramienta Pencil"*

[18] ***Pencil*** se trata de una herramienta de creación de prototipos de GUI de código abierto y multiplataforma.

Elección

Se ha escogido esta herramienta debido a que ya me encontraba familiarizado con ella, gracias a su uso durante el transcurso del grado.

Estudios realizados

Para poder llevar a cabo el desarrollo de dicho proyecto, previamente he tenido que analizar con detenimiento las facturas mensuales que reciben en la comunidad de vecinos, para de esta forma entender de donde provienen los importes totales, ya que estos dependen de importes fijos y variables.

También me ha sido necesario estudiar todo lo necesarios para levantar una aplicación *Django* en un servidor *CentOs*, ya que anteriormente prácticamente carecía de conocimientos respecto a este tema.

Por último, aunque como he comentado previamente en el grado ya había utilizado el *framework Django*, también me ha sido necesario repasarme toda su documentación oficial para de esta forma saber todas las posibilidades que este *framework* me ofrece y explotarlas según la necesidad de cada funcionalidad.

Capítulo 3: Análisis

En este capítulo se detallarán las diferentes fases por las que se ha pasado para poder realizar un correcto desarrollo, pasando inicialmente por el *Análisis de los requerimientos* siguiendo por el *Análisis del diseño de las funcionalidades*, *Modelo de dominio* e *Interfaz*.

Análisis de Requerimientos

- *Requerimientos Funcionales*

Seguidamente se enumeran los requerimientos funcionales de la aplicación, junto con una pequeña descripción:

Requerimientos Funcionales	Descripción
Requerimento 1	La aplicación debe permitir añadir facturas mensuales a un usuario administrador.
Requerimento 2	La aplicación debe permitir añadir lecturas mensuales a un usuario administrador.
Requerimento 3	La aplicación debe permitir añadir el fichero .csv de estadísticas mensuales a un usuario administrador.
Requerimento 4	La aplicación debe permitir añadir factuas bimensuales a un usuario administrador.
Requerimento 5	La aplicación debe permitir la gestión de la cuenta de usuario a cualquier usuario.
Requerimento 6	La aplicación debe permitir consultar los consumos mensuales por escalera a cualquier usuario.
Requerimento 7	La aplicación debe permitir consultar las facturas mensuales por bloque a cualquier usuario.
Requerimento 8	La aplicación debe permitir consultar los totales mensuales por bloque a cualquier usuario.
Requerimento 9	La aplicación debe permitir consultar mediante un gráfico, la comparativa del "% consumo ACS entre el Bloque 1 y el Bloque 2" a cualquier usuario.
Requerimento 10	La aplicación debe permitir consultar mediante un gráfico, la comparativa del "% consumo CAL entre el Bloque 1 y el Bloque 2" a cualquier usuario.
Requerimento 11	La aplicación debe permitir consultar mediante un gráfico, la comparativa del "% consumo ACS vs % consumo CAL" por bloque a cualquier usuario.
Requerimento 12	La aplicación debe permitir consultar el histórico de facturas mensuales por bloques a cualquier usuario.
Requerimento 13	La aplicación debe permitir consultar el histórico de lecturas mensuales por bloque a cualquier usuario.
Requerimento 14	La aplicación debe permitir añadir un fichero .csv con la finalidad de crear el mapeo "usuario-piso" a un usuario administrador.
Requerimiento 15	La aplicación debe permitir una gestión de mensajes, a través de la cual se mantendrá informado a cada usuario de las diferentes noticias de la misma.

- *Requerimientos no funcionales*

Seguidamente se enumeran los requerimientos no funcionales de la aplicación, junto con una pequeña descripción:

Requerimientos No Funcionales	Descripción
Requerimento 1	El servidor en el que se realice el despliegue de producción, debe disponer del sistema operativo CentOS.
Requerimento 2	El servidor en el que se realice el despliegue de producción, debe disponer de conexión a internet.
Requerimento 3	El servidor en el que se realice el despliegue de producción, debe disponer de un motor de base de datos PostgreSQL.
Requerimento 4	El servidor que sirve la aplicación, debe de disponer de conexión a internet.
Requerimento 5	El dispositivo que acceda a la aplicación debe disponer de conexión a internet.
Requerimento 6	En caso de tratarse de un dispositivo Smartphone o Tablet, debe disponer del sistema operativo Androd xx.xx
Requerimento 7	La aplicación debe proporcionar unos tiempos de consulta óptimos.
Requerimento 8	La aplicación debe guardar de forma segura los datos de la comunidad de vecinos.

Análisis de Diseño

En este apartado se presentarán los diferentes análisis de diseño realizados para la construcción del aplicativo web.

- *Análisis del diseño del modelo de dominio*

Motor de base de datos

El uso del *framework Django* para el desarrollo de la aplicación, ha centrado mi elección del motor de base de datos entre *MySQL* y *PostgreSQL*, ya que estos son dos de los tres motores que soporta.
Sqlite3, es el tercer motor con el que el *framework* tiene compatibilidad, pero no es correcta su utilización en un despliegue de producción ya que únicamente permite un acceso secuencial de datos.

Una vez cerrado el círculo de elección, el motivo por el cual me ha hecho decidir por trabajar con *PostgreSQL* es su *Licencia MIT (Software libre)* a diferencia de *MySQL* que está basado en una *Licencia GPL* o *Licencia Comercial* en manos de *ORACLE*.

Modelo de dominio

En este apartado se desglosan las diferentes entidades, junto con sus respectivos campos, de las que consta el modelo de dominio de la aplicación.

"UploadsController"

Esta entidad consta de un único registro, el cual contiene la información de todas las subidas de los registros del último mes en la aplicación:

- *Identificador*: Es el campo que forma la clave primaria, el cual tiene como valor estático "uploads".
- *csv_file_subido1*: Guarda la fecha de la última subida del fichero .csv del bloque 1.
- *csv_file_subido2*: Guarda la fecha de la última subida del fichero .csv del bloque 2.
- *lectura_subida_1*: Guarda la fecha de la última subida de la lectura del bloque 1.
- *lectura_subida_2*: Guarda la fecha de la última subida de la lectura del bloque 2.
- *factura_subida_gas_1*: Guarda la fecha de la última subida de la factura de gas del bloque 1.
- *factura_subida_agua_1*: Guarda la fecha de la última subida de la factura de agua del bloque 1.
- *factura_subida_luz_1*: Guarda la fecha de la última subida de la factura de luz del bloque 1.
- *factura_subida_gas_2*: Guarda la fecha de la última subida de la factura de gas del bloque 2.
- *factura_subida_agua_2*: Guarda la fecha de la última subida de la factura de agua del bloque 2.
- *factura_subida_luz_2*: Guarda la fecha de la última subida de la factura de luz del bloque 2.
- *ultima_subida*: Guarda la fecha de la última subida mensual.

"CSVFiles"

Esta entidad es la encargada de almacenar los archivos estadísticos .csv de forma temporal hasta realizar el cálculo final, una vez realizado el cálculo final, se borran todos los registros:

- *filename*: Guarda el nombre del archivo.
- *docfile*: Guarda el fichero .csv , apunta al directorio '/media/csv/'.
- *periodo*: Guarda el período al que pertenece el fichero.
- *numero_bloque*: Este campo forma la clave primaria y guarda el número de bloque al que pertenece el fichero.

"FacturesMensualsTemporals"

Esta entidad se encarga de almacenar de forma temporal todas las facturas de la subida mensual, con la finalidad de retenerlas y evitar errores, hasta realizar el cálculo final:

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *num_factura*: Guarda el número de la factura correspondiente.
- *tipo*: Guarda el tipo de factura (Gas, Agua o Luz).
- *periodo*: Guarda el período al que corresponde la factura.
- *fijo*: Guarda el importe fijo de la factura.
- *variable*: Guarda el importe variable de la factura.
- *numero_bloque*: Guarda el número de bloque al que pertenece la factura.
- *filename*: Guarda el nombre del fichero .pdf de la factura.
- *docfile*: Guarda el fichero .pdf, apunta al directorio '/media/pdf/'.
- *mes*: Guarda el mes al que pertenece la factura.

Combinación campos para clave secundaria: *tipo* y *numero_bloque*.

"LecturesMensualsTemporals"

Esta entidad se encarga de almacenar de forma temporal todas las lecturas de la subida mensual, con la finalidad de retenerlas y evitar errores, hasta realizar el cálculo final:

- *general_actual*: Guarda el valor del consumo general de la lectura.
- *numero_bloque*: Este valor forma la clave primaria y guarda el número de bloque al que pertenece la lectura.
- *acs_actual*: Guarda el valor del consumo ACS de la lectura.
- *periodo*: Guarda el período al que corresponde la lectura.
- *mes*: Guarda el mes al que pertenece la lectura.

"Mapping"

Esta entidad se encarga de almacenar el mapeo de cada usuario con su respectivo identificador de piso:

- *id_piso*: Este campo forma la clave primaria y guarda el identificador de cada piso.
- *piso*: Guarda el valor referente al número de piso (Ej. 1ero 2ª).
- *bloque*: Guarda el número del bloque al que pertenece el registro.
- *escalera*: Guarda el número de la escalera al que pertenece el registro.

"Mensaje"

Esta entidad se encarga de almacenar todos los registros de las notificaciones enviadas en la aplicación *(Funcionalidad para la versión 2)*:

- *id*: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *titulo*: Guarda el título del mensaje.
- *asunto*: Guarda el asunto del mensaje.
- *mensaje*: Guarda el cuerpo del mensaje.
- *user*: Este campo forma la clave foránea y guarda un registro de la entidad "user" al que pertenece el mensaje.
- *visto*: Guarda verdadero o falso, dependiendo de sí el mensaje se ha leído o no.
- *fecha*: Guarda la fecha en la que se ha emitido el mensaje.

"Usuario"

Esta entidad se encarga de almacenar los datos de cada usuario y a la vez extiende la entidad "User" del sistema, para permitir una mayor gestión de la cuenta de usuario:

- *id*: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *user*: Este campo forma la clave foránea y guarda un registro de la entidad "user" con la que se relaciona.
- *password_key*: Guarda el valor de la contraseña del usuario.
- *block*: Guarda el valor del bloque al que pertenece el usuario.
- *piso*: Guarda el piso al que pertenece el usuario.
- *fecha_registro*: Guarda la fecha en la que se creó la cuenta del usuario correspondiente.
- *id_piso*: Guarda el identificador del piso que pertenece al usuario.

"RegistrosGráficos"

Esta entidad se encarga de almacenar los registros que posteriormente se mostrarán en el apartado de los gráficos:

- *periodo*: Guarda el período al que pertenece el registro.
- *consumo_acs*: Guarda el valor del consumo de ACS.
- *consumo_cal*: Guarda el valor del consumo de CAL.
- *numero_bloque*: Guarda el número de bloque al que pertenece cada registro.

"FacturesHistorics"

Esta entidad se encarga de almacenar todas las facturas del aplicativo, una vez ha pasado su período:

- *id*: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *num_factura*: Guarda el número de la factura correspondiente.
- *tipo*: Guarda el tipo de factura al que corresponde la factura (Gas, Agua o Luz).
- *periodo*: Guarda el período al que pertence la factura.
- *fijo*: Guarda el importe fijo de la factura.
- *variable*: Guarda el importe variable de la factura.
- *total_factura*: Guarda el importe total de la factura.
- *numero_bloque*: Guarda el número de bloque al que pertenece la factura.
- *filename*: Guarda el nombre del fichero .pdf de la factura.
- *docfile*: Guarda el fichero .pdf, apunta al directorio '/media/pdf/'.
- *perdues*: Guarda el valor de las pérdidas de la factura.
- *perdues_eur*: Guarda el valor en Euros (€) de las pérdidas.
- *fixe_perdues*: Guarda el valor fijo de las pérdidas.
- *variable_perdues*: Guarda el valor variable de las pérdidas.

Combinación campos para clave secundaria: *tipo*, *período* y *numero_bloque*.

"LecturesMensualsHistorics"

Esta entidad se encarga de almacenar todas las lecturas del aplicativo, una vez ha pasado su período:

- *id*: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *general_actual*: Guarda el valor del consumo general para el período actual.
- *numero_bloque*: Guarda el número de bloque al que corresponde la lectura.
- *general_anterior*: Guarda el valor del consumo general para el período anterior.
- *consum_general*: Guarda el valor resultado de "*general_actual – general_anterior*" .
- *acs_actual*: Guarda el valor del consumo ACS para el período actual.
- *acs_anterior*: Guarda el valor del consumo ACS para el período anterior.
- *consum_acs*: Guarda el valor resultado de "*acs_actual – acs_anterior*".
- *cal_actual*: Guarda el valor del consumo CAL para el período actual.
- *cal_anterior*: Guarda el valor del consumo CAL para el período anterior.
- *consum_cal*: Guarda el valor resultado de "*cal_actual – cal_anterior*".
- *perdues*: Guarda el valor de las pérdidas para la lectura correspondiente.
- *perdues_percent*: Guarda el % de las pérdidas para la lectura correspondiente.
- *acs_percent*: Guarda el % del consumo ACS.
- *cal_percent*: Guarda el % del consumo CAL.
- *periodo*: Guarda el período al que corresponde la lectura correspondiente.

"ConsumsGeneralsMensualsHistoricos"

Esta entidad se encarga de almacenar todos los registros referentes a la tabla "Consumos Generales Mensuales" una vez ha pasada su mensualidad:

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *comptador_general*: Guarda el valor del contador general, para el bloque correspondiente.
- *suma_veins_cal*: Guarda el valor de la suma del consum CAL para todos los vecinos del bloque correspondiente.
- *suma_veins_acs*: Guarda el valor de la suma del consum ACS para todos los vecinos del bloque correspondiente.
- *total_cal_acs*: Guarda el valor resultado de "*suma_veins_acs* + *suma_veins_cal*"
- *cal_percent_consumit*: Guarda el % del consumo de CAL para el bloque correspondiente.
- *acs_percent_consumit*: Guarda el % de consumo de ACS para el bloque correspondiente.
- *numero_bloque*: Guarda el número de bloque al que pertence el registro.
- *periodo*: Guarda el período al que pertenece el registro.

"TotalsFacturesMensualsHistoricos"

Esta entidad se encarga de almacenar la tabla con los totales de las facturas mensuales para cada bloque, una vez ha pasado su mensualidad:

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *total_factures*: Guarda el valor total de la factura correspondiente.
- *cal_fixe*: Guarda el importe fijo correspondiente del CAL de la factura.
- *cal_variable*: Guarda el importe variable del CAL de la factura.
- *acs_fixe*: Guarda el importe fijo del ACS de la factura.
- *acs_variable*: Guarda el importe variable del ACS de la factura.
- *tipo*: Guarda el tipo al que pertenece la factura (Gas, Agua o Luz).
- *numero_bloque*: Guarda el número de bloque al que pertenece la factura.
- *periodo*: Guarda el período al que pertenece la factura.

"ValoresEstadísticos"

Esta entidad se encarga de almacenar los registros obtenidos de la lectura del fichero estadístico .csv, guardando de esta forma un histórico de todas las estadísticas.

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *numero*: Guarda el valor 6 o 16, refiriéndose a ACS o CAL.
- *valor*: Guarda el valor de consumo.
- *unidad*: Guarda la unidad en la que se mide el consumo del registro.
- *fecha*: Guarda la fecha a la que pertenece el registro.
- *id_piso*: Guarda el identificador del piso al que pertenece el registro.

Combinación campos para clave secundaria: *numero*, *fecha* y *id_piso*.

"FacturesMensuals"

Esta entidad se encarga de almacenar durante cada mensualidad, las facturas correspondientes, y una vez se cambia de mensualidad se mueven a la entidad "*FacturesHistorics*":

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *num_factura*: Guarda el número de la factura correspondiente.
- *tipo*: Guarda el tipo al que pertenece la factura (Gas, Agua o Luz).
- *periodo*: Guarda el período al que pertence la factura.
- *fijo*: Guarda el importe fijo de la factura.
- *variable*: Guarda el importe variable de la factura.
- *total_factura*: Guarda el importe total de la factura.
- *numero_bloque*: Guarda el número de bloque al que pertenece la factura.
- *filename*: Guarda el nombre del fichero .pdf de la factura.
- *docfile*: Guarda el fichero .pdf de la factura, apunta a '/media/pdf/'.
- *perdues*: Guarda el valor de las pérdidas de la factura correspondiente.
- *perdues_eur*: Guarda el valor en Euros (€) de la factura correspondiente.
- *fixe_perdues*: Guarda el valor fijo de las pérdidas de la factura correspondiente.
- *variable_perdues*: Guarda el valor variable de las pérdidas de la factura correspondiente.

Combinación campos para clave secundaria: *tipo* y *número_bloque.*

"LecturesMensuals"

Esta entidad se encarga de almacenar para cada mensualidad las lecturas correspondientes, una vez ha pasado la mensualidad, estas lecturas se trasladan a la entidad "*LecturesMensualsHistorics*":

- *general_actual*: Guarda el valor del consumo general para el período actual.
- *numero_bloque*: Este campo forma la clave primaria y guarda el número de bloque al que corresponde la lectura.
- *general_anterior*: Guarda el valor del consumo general para el período anterior.
- *consum_general*: Guarda el valor resultado de "*general_actual – general_anterior*" .
- *acs_actual*: Guarda el valor del consumo ACS para el período actual.
- *acs_anterior*: Guarda el valor del consumo ACS para el período anterior.
- *consum_acs*: Guarda el valor resultado de "*acs_actual – acs_anterior*".
- *cal_actual*: Guarda el valor del consumo CAL para el período actual.
- *cal_anterior*: Guarda el valor del consumo CAL para el período anterior.
- *consum_cal*: Guarda el valor resultado de "*cal_actual – cal_anterior*".
- *perdues*: Guarda el valor de las pérdidas para la lectura correspondiente.
- *perdues_percent*: Guarda el % de las pérdidas para la lectura correspondiente.

- *acs_percent*: Guarda el % del consumo ACS.
- *cal_percent*: Guarda el % del consumo CAL.
- *periodo*: Guarda el período al que corresponde la lectura correspondiente.

"ConsumsGeneralsMensuals"

Esta entidad se encarga de almacenar todos los registros referentes a la tabla "Consumos Generales Mensuales" para la mensualidad actual, una vez esta ha pasado, los registros son trasladados a la entidad "*ConsumsGeneralsHistorics*":

- id: Este campo forma la clave primaria y guarda un identificador de cada registro.
- *comptador_general*: Guarda el valor del comptador general para el bloque correspondiente.
- *suma_veins_cal*: Guarda el valor de la suma del consum CAL para todos los vecinos del bloque correspondiente.
- *suma_veins_acs*: Guarda el valor de la suma del consum ACS para todos los vecinos del bloque correspondiente.
- *total_cal_acs*: Guarda el valor resultado de "*suma_veins_acs* + *suma_veins_cal*"
- *cal_percent_consumit*: Guarda el % del consumo de CAL para el bloque correspondiente.
- *acs_percent_consumit*: Guarda el % de consumo de ACS para el bloque correspondiente.
- *numero_bloque*: Guarda el número de bloque al que pertence el registro.
- *periodo*: Guarda el período al que pertenece el registro.
- *suma_veins_m_acs:* Guarda el valor del resultado de la suma del consumo ACS para todos los vecinos del bloque correspondiente.

Combinación campos para clave secundaria: *tipo* y *número_bloque.*

Análisis del diseño de las funcionalidades

Previamente al desarrollo de las funcionalidades, se realizó un análisis exhaustivo de estas para de esta forma conseguir evitar errores innecesarios durante el desarrollo.
Este análisis consiste en un diagrama de interacción para cada funcionalidad, estudiando el comportamiento que el sistema y la aplicación deben llevar durante la ejecución de dichas funcionalidades.

Los diagramas de interacción, han sido creados mediante la herramienta online libre *Cacoo*.

Requerimiento funcional 1: Añadir facturas mensuales i Requerimiento funcional 4: Añadir facturas bimensuales.

Esta funcionalidad tiene como finalidad permitir, a los usuarios administrador del sistema de la aplicación, añadir las facturas correspondientes a cada mensualidad y bloque.

Se inicia en el momento de realizar la petición *GET* hacía la página *Añadir Facturas Mensuales* antes de que esta se le muestre al usuario, se comprueba si anteriormente se han subido las lecturas y los ficheros correspondientes a la mensualidad, en caso contrario devuelve una página de error 404 indicando cuales son los pasos necesarios a realizar antes de poder realizar la subida de las facturas.

En el caso de la llamada *POST*, siempre se comprueba que no se haya subido para la mensualidad correspondiente una factura del mismo tipo y número de bloque, en el caso de ya haberse subido, se devuelve un mensaje de alerta indicando que ya existe una factura para el tipo y número de bloque en el período correspondiente, en caso contrario esta se sube a la entidad *FacturesMensualsTemporals*.

En el caso del *Requerimiento Funcional 4: Añadir facturas bimensuales*, el diagrama de interacción actúa de la misma manera que para las facturas mensuales.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Añadir Facturas']

Requerimiento funcional 2: Añadir lecturas mensuales

Esta funcionalidad tiene como finalidad, permitir a los usuarios administradores del sistema de la aplicación, añadir las lecturas correspondientes a cada mensualidad y bloque.

Se inicia en el momento de realizar la petición *GET* hacía la página *Añadir Lecturas Mensuales*, antes de que esta se le muestre al usuario, se comprueba que previamente se hayan subido los ficheros .csv correspondientes a la mensualidad, en caso contrario devuelve una página de error 404 indicando cuales son los pasos necesarios a realizar antes de poder realizar la subida de las lecturas.

En el caso de la llamada *POST*, siempre se comprueba que no se haya subido para la mensualidad correspondiente una lectura para el mismo número de bloque, en el caso de ya haberse subido, se devuelve un mensaje de alerta indicando que ya existe una lectura para el número de bloque en el período correspondiente, en caso contrario esta se sube a la entidad *LecturesMensualsTemporals*.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Añadir Lecturas']

Requerimiento funcional 3: Añadir '.csv' mensuales

Esta funcionalidad tiene como finalidad, permitir a los usuarios administradores del sistema de la aplicación, añadir los ficheros .csv correspondientes a cada mensualidad y bloque.
Se inicia en el momento de realizar la petición *GET* hacía la página *Subir Fichero .CSV* no se realiza ninguna comprobación previa y se muestra directamente al usuario.

En el caso de la llamada *POST*, siempre se comprueba que no se haya subido para la mensualidad correspondiente un fichero .csv para el mismo número de bloque, en el caso de ya haberse subido, se devuelve un mensaje de alerta indicando que ya existe un fichero .csv para el número de bloque en el período correspondiente, en caso contrario esta se sube a la entidad *CSVFiles*.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Añadir Fichero .CSV']

Requerimiento funcional 5: Gestionar cuenta usuario

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación, poder gestionar su contraseña. Inicialmente se pensó en permitirle poder gestionar toda la información que le pertenece, pero después de analizarlo se llegó a la conclusión que los dos aspectos realmente necesarios para el uso de la aplicación son la contraseña y el id del piso, donde este último quedará a cargo del usuario administrador modificarlo si es necesario.

Se inicia en el momento de que el usuario se dispone a cambiar la contraseña, primero se comprueba que su actual contraseña sea la correcta y que la nueva coincida en los dos casos y en caso afirmativo se procede al cambio de la misma. En el caso de la contraseña actual no coincidir o la nueva no ser la misma en los dos casos se devuelve a la pantalla de cambio de contraseña con un mensaje alertando del error correspondiente.

Anexo 1:
[Diagrama de interacción del requerimiento funcional 'Gestionar Cuenta Usuario']

Requerimiento funcional 6: Consultar consumos mensuales

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar su consumo mensual a la par que los de todos los otros vecinos.
Se inicia cuando el usuario hace la petición para obtener el consumo mensual de su escalera, ya que los consumos se agrupan por escaleras, el sistema se encarga de crear una nueva instancia de Escalera la cual mediante unos cálculos que más adelante se comentarán, se encarga de devolver todos los consumos de la escalera solicitada.

Anexo 1:
[Diagrama de interacción del requerimiento funcional 'Consultar Consumos Mensuales]

Requerimiento funcional 7: Consultar facturas mensuales

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar las facturas mensuales (Gas, Agua y Luz) del bloque correspondiente.

Se inicia cuando el usuario realiza la petición para obtener las facturas de su bloque, el sistema crea una nueva instancia de *FacturasBloque*, la cual se encarga de realizar unos cálculos, que más adelante se explicarán, y devolver la lista de facturas para el bloque correspondiente.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Consultar Facturas Mensuales']

Requerimiento funcional 8: Consultar totales mensuales

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar las tablas de totales mensuales del bloque correspondiente.

Se inicia cuando el usuario realiza la petición para obtener los totales, el sistema hace una consulta a la base de datos a la entidad *TotalsFacturesMensuals* filtrando por el número de bloque correspondiente, junto con otra a la entidad *ConsumsGeneralsMensuals*, devolviendo de esta forma la lista de totales y consumos, para el bloque correspondiente.

Anexo 1:
[Diagrama de interacción del requerimiento funcional 'Consultar Totales Mensuales']

Requerimiento funcional 9: Consultar gráfico "% Consumo ACS Bloque 1 vs Bloque 2"

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar una comparativa del consumo ACS en % entre el bloque 1 y el bloque 2, durante el último año, mediante una gráfica.

Se inicia cuando el usuario realiza la petición para obtener la comparativa, el sistema se encarga de obtener los registros ACS, mediante una llamada a la base de datos, renderizando estos datos en una gráfica.

Anexo 1: ['Consultar Gráfico % Consumo ACS Bloque 1 vs Bloque 2']

Requerimiento funcional 10: Consultar gráfico "% Consumo CAL Bloque 1 vs Bloque 2"

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar una comparativa del consumo CAL en % entre el bloque 1 y el bloque 2, durante el último año, mediante una gráfica.

Se inicia cuando el usuario realiza la petición para obtener la comparativa, el sistema se encarga de obtener los registros CAL, mediante una llamada a la base de datos, renderizando estos datos en una gráfica.

Anexo 1: ['Consultar Gráfico % Consumo CAL Bloque 1 vs Bloque 2']

Requerimiento funcional 11: Consultar gráfico "% Consumo ACS vs % Consumo CAL"por bloque

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar una comparativa del consumo ACS en % contra el consumo CAL en %, para cada bloque.

Se inicia cuando el usuario realiza la petición para obtener la comparativa, el sistema se encarga de obtener los datos comparativos, mediante una llamada a la base de datos, renderizando estos datos en una gráfica.

Anexo 1: ['Consultar Gráfica % Cons. ACS vs Cons. CAL (por bloque)']

Requerimiento funcional 12: Consultar histórico de facturas

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación poder consultar el histórico total de facturas para cada bloque.

Se inicia cuando el usuario realiza la petición para obtener el histórico de facturas, el sistema hace una llamada a la base de datos filtrando la entidad *FacturesHistorics* a través del número de bloque correspondiente, y se le devuelve un listado con todas las facturas del bloque.

Anexo 1:
[Diagrama de interacción del requerimiento funcional 'Consultar Histórico Facturas']

Requerimiento funcional 13: Consultar histórico lecturas

Esta funcionalidad tiene como finalidad, permitir a cualquier usuario de la aplicación consultar el histórico de todas las lecturas mensuales para el bloque correspondiente.

Se inicia cuando el usuario realiza la petición para obtener el histórico de lecturas, el sistema hace una llamada a la base de datos filtrando la entidad *LecturasMensualsHistorics* a través del número de bloque correspondiente, y se le devuelve un listado con todas las lecturas del bloque.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Consultar Histórico Lecturas']

Requerimiento funcional 14: Añadir fichero .csv de mapeo

Esta funcionalidad tiene como finalidad, permitir al usuario administrador añadir un fichero .csv que contenga la relación entre el *id_piso* y el piso real, para permitir de esta forma a la aplicación tener relacionados todos los *id_piso* con el piso correspondiente.

Se inicia cuando el usuario administrador realiza una inserción del fichero .csv que contiene el mapeo de pisos-usuario, el sistema se encarga de realizar una llamada a la base de datos insertando todos los registros relacionales.

Anexo 1: [Diagrama de interacción del requerimiento funcional 'Añadir Fichero .CSV']

- *Análisis del diseño de la interfaz*

Previamente al desarrollo de la interfaz de la aplicación, se realizó un análisis de cómo esta se tiene que presentar al usuario, para minimizar los riesgos de frustración en el uso de la misma y de esta forma desarrollar una aplicación usable.
Para ello se han realizado unos bocetos de cada una de las pantallas a las que el usuario tendrá acceso, mediante el software libre *Pencil*.

"Pantalla de inicio de sesión"

Dicha pantalla está basada en un diseño sencillo, poco cargado e intuitivo, ya que considero que al tratarse de la primera pantalla de la aplicación a la que el usuario se expone debe de estar lo menos cargada posible y resultar ser sencilla para que de esta forma su experiencia inicie con buen pie.

- *Introducir nombre usuario*: Cuadro de texto en el que se ha de introducir el nombre de usuario.
- *Introducir contraseña*: Cuadro de texto en el que se ha de introducir la contraseña.

- *Recordar contraseña*: CheckBox que permite recordar las credenciales en el navegador.
- *¿Ha olvidado su contraseña?*: Enlace a la página de recuperación de contraseña.
- *Submit*: Botón para proceder a iniciar sesión.

Anexo 2: [Mockup de la pantalla destinada al 'Inicio de Sesión']

"Pantalla menú principal"

El diseño del menú principal sigue el mismo criterio comentado anteriormente, en el cual se puede observar una barra lateral sencilla, un pase de diapositivas y un cuadro de texto informativo.

- *Barra lateral*: Contiene el acceso a todas las secciones de la aplicación.
- *Imagen central*: Pase de diapositivas con imágenes de la comunidad de vecinos.
- *Cuadro de texto*: Contiene una breve explicación de la finalidad de la aplicación junto con datos de contacto con el creador de la misma, para futuras consultas.
- *Button*: Hace referencia al botón encargado de cerrar la sesión.
- *Imagen de un sobre*: Enlace a la página que contiene las notificaciones de la aplicación.

Anexo 2: [Mockup de la pantalla destinada al 'Menú Principal']

"Pantalla de consulta de datos en una única tabla"

El diseño para la consulta de datos, ya sean históricos o mensuales, ha sido el mismo a través de una única tabla, ya que de esta forma no se generará desconcierto al usuario y se le permitirá familiarizarse más con la aplicación.

Hace referencia, a todas las pantallas de consulta de históricos y mensuales, menos la consulta de los totales mensuales, ya que está varia en cierta forma.

- *Tabla de datos*: En la tabla se presenta la información de la consulta.
- *Botones next y prev.*: Botones encargados de mostrar los siguientes o anteriores registros en la tabla.

*Anexo 2: [*Mockup 'Consulta Datos Única Tabla'*]*

"Pantalla de consulta de datos en dos tablas"

El diseño para la consulta de los totales mensuales, ha sido pensado según como anteriormente se les presentaba a los usuarios mediante el Excel, ya que se les mostraba en dos tablas donde una hacía referencia a los totales mensuales de las facturas y la otra a los totales de los consumos mensuales.
De esta forma para no cambiar la forma de presentárselo y seguir manteniendo la sensación de familiarización del usuario con la aplicación, se ha decidido presentar en una única vista las dos tablas mencionadas.

- *Tabla de datos 1*: Se presenta la información sobre la consulta de los totales de las facturas mensuales.
- *Tabla de datos 2*: Se presenta la información sobre la consulta de los totales de los consumos mensuales.

*Anexo 2: [*Mockup 'Consulta Datos Dos Tablas'*]*

"Pantallas administración de datos"

El diseño de dichas pantallas, sigue manteniendo la línea seguida hasta el momento, presentando al usuario administrador un único formulario de inserción de datos sencillo para cada tipo de estos, facilitando de esta forma en todo momento al usuario su experiencia en la aplicación y tenerlo en todo momento a sabiendas del que está subiendo y porque lo está haciendo.

"Añadir fichero .csv estadístico"

- *Drop files here*: Botón para adjuntar el fichero .csv.
- *Número bloque*: Desplegable para permitir seleccionar el bloque correspondiente.
- *Período*: Desplegable para permitir seleccionar el mes correspondiente al período.

*Anexo 2: [*Mockup 'Añadir Fichero .CSV Estadístico'*]*

"Añadir lecturas mensuales"

- *Consumo general*: Cuadro numérico en el que se ha de introducir el consumo general.
- *ACS Actual*: Cuadro numérico en el que se ha de introducir el consumo ACS.
- *Número bloque*: Desplegable para permitir seleccionar el bloque correspondiente.
- *Período*: Desplegable para permitir seleccionar el mes correspondiente al período.

*Anexo 2: [*Mockup 'Añadir Lecturas Mensuales'*]*

"Añadir facturas mensuales"

- *Nº factura*: Cuadro de texto en el que se ha de introducir el número de la factura.
- *Número bloque*: Desplegable para permitir seleccionar el bloque correspondiente.

- *Período*: Desplegable para permitir seleccionar el mes correspondiente al período.
- *Importe fijo*: Cuadro numérico en el que se ha de introducir el importe fijo de la factura.
- *Importe variable*: Cuadro numérico en el que se ha de introducir el importe variable de la factura.
- *Drop files here*: Botón para adjuntar el archivo .pdf de la factura.

Anexo 2: [Mockup 'Añadir Facturas Mensuales']

"Añadir facturas bimensuales"

Se ha decidido crear una página diferente para las facturas bimensuales, ya que como estas no necesitan la inserción de datos, facilitar de esta forma la tarea del administrador y únicamente tener que indicar el número de bloque, el período y el tipo de factura al que pertenece. Agilizando de esta forma la subida de datos.

- *Número bloque*: Desplegable para permitir seleccionar el bloque correspondiente.
- *Período*: Desplegable para permitir seleccionar el mes correspondiente al período.
- *Tipo Factura*: Desplegable para permitir seleccionar el tipo al que la factura bimensual corresponde.

*Anexo 2: [*Mockup 'Añadir Facturas BiMensuales'*]*

"Añadir fichero .csv de mapping"

- *Drop files here*: Botón para adjuntar el archivo .csv con el mapping.

*Anexo 2: [*Mockup 'Añadir Fichero .CSV Mapping'*]*

“Pantalla gestión contraseña usuario”

Manteniendo el criterio seguido durante el diseño de la interfaz de toda la aplicación, también se ha decidido realizar una pantalla muy sencilla e intuitiva para que el usuario pueda gestionar su contraseña sin ocasionarle ningún tipo de problema.

- *Contraseña actual*: Cuadro de texto en el que se ha de introducir la contraseña actual.
- *Nueva contraseña*: Cuadro de texto en el que se ha de introducir la nueva contraseña.
- *Repetir nueva contraseña*: Cuadro de texto en el que se ha de volver a introducir la nueva contraseña.

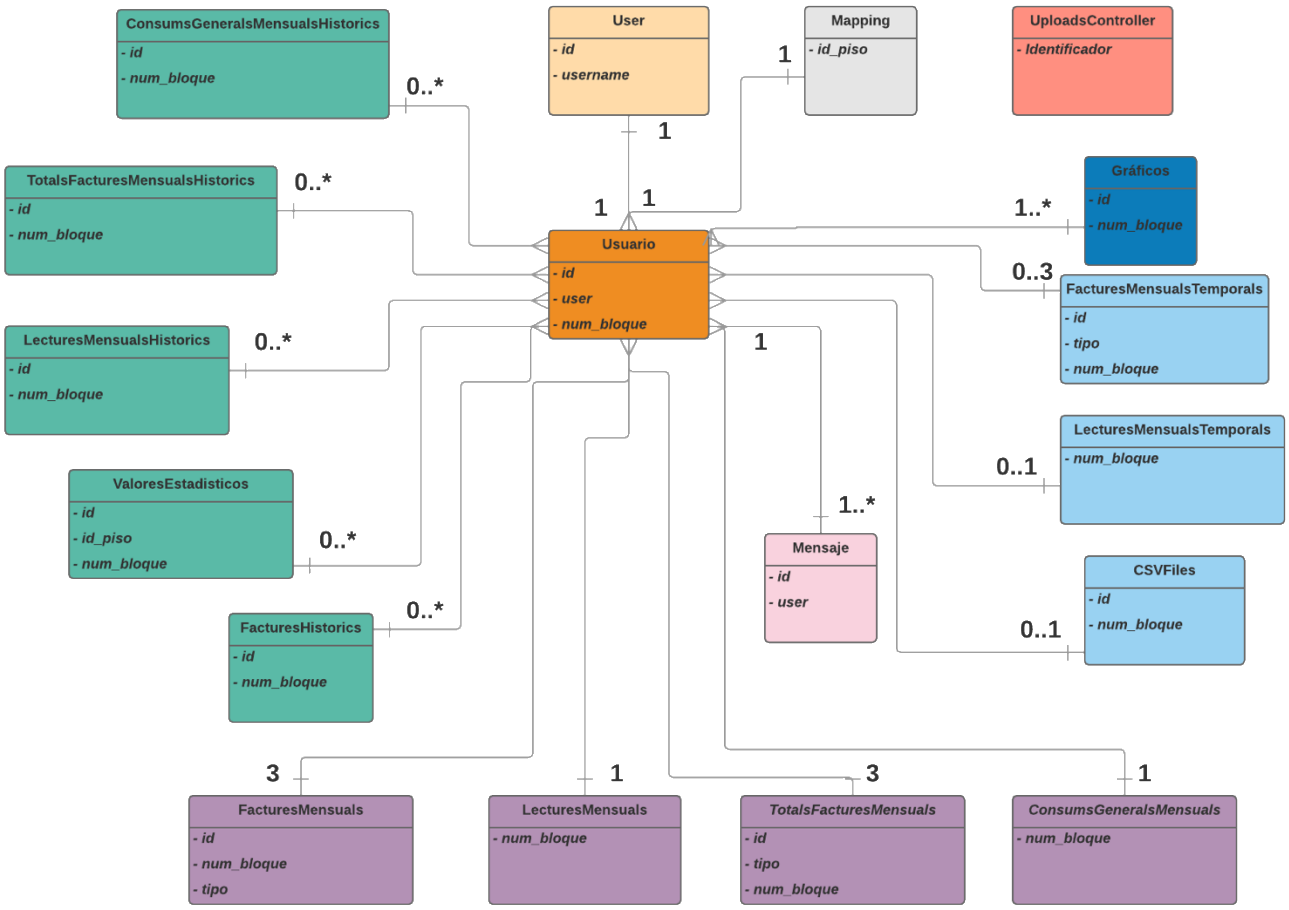
*Anexo 2: [*Mockup ‘Gestión Contraseña Usuario’*]*

Capítulo 4: Desarrollo

En este capítulo, una vez ya hecho el análisis, se intentarán plasmar los diferentes criterios seguidos para llevar a cabo el desarrollo de la aplicación en los apartados de modelo de dominio, requerimientos funcionales e interfaz.

Desarrollo del modelo de dominio

Para representar gráficamente la relación entre las diferentes entidades de la aplicación, se ha realizado un diagrama UML de clases, para ver con más claridad con que entidades está relacionada la entidad usuario, ya que se trata de la entidad principal, y qué valor tiene está relación.



Una vez presentado el diagrama UML, se explicarán cada una de las relaciones a las que pertenece la entidad “Usuario”, pero antes de esto, es necesario añadir que el bloque 1 dispone de 23 pisos y el bloque 2 de 24 pisos, donde cada piso corresponde a un registro de la entidad “Usuario”.

- *Entidades de registros históricos*

Las relaciones que se describirán a continuación, están basadas con los registros históricos de la comunidad, por lo que estos siempre irán de 0 a infinito, 0 porque al arranque de la aplicación aún no habrá ningún registro histórico en las entidades, e infinitos porque se trata de una acumulación de registros creciente de forma mensual:

- "*Usuario – ConsumsGeneralsHistorics*": La entidad "*ConsumsGeneralsHistórics*" tiene los registros diferenciados por número de bloque por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 0...* registros de la entidad "*ConsumsGeneralsHistorics*".

- "*Usuario - TotalsFacturesMensualsHistorics*": La entidad "*TotalsFacturesMensualsHistorics*" tiene los registros diferenciados por número de bloque por lo que podemos afirmar que cada 23…24 registros de la entidad "*Usuario*" están relacionados con 0…* registros de la entidad "*TotalsFacturesMensualsHistorics*".

- "*Usuario – LecturesMensualsHistorics*": La entidad "*LecturesMensualsHistorics*" tiene los registros diferenciados por número de bloque por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 0..* registros de la entidad "*LecturesMensualsHistorics*".

- "*Usuario – FacturesMensualsHistorics*": La entidad "*FacturesMensualsHistorics*" tiene los registros diferenciados por número de bloque por lo que podemos afirmar que cada 23…24 registros de la entidad "*Usuario*" están relacionados con 0…* registros de la entidad "*FacturesMensualsHistorics*".

- "*Usuario – ValoresEstadísticos*": La entidad "*ValoresEstadisticos*" tiene los registros diferenciados por número de bloque por lo que podemos afirmar que cada 23…24 registros de la entidad "*Usuario*" están relacionados con 0...* registros de la entidad "*ValoresEstadisticos*".

- *Entidades de registros mensuales*

Las relaciones que se describirán a continuación, están basadas en los registros mensuales de la comunidad, donde damos por supuesto que desde el arranque de la aplicación ya contendrán la información correspondiente a la mensualidad y por lo tanto dichas entidades nunca se encontrarán vacías.
Es necesario añadir también que, en cada cambio de mensualidad, los registros serán trasladados a las entidades de registros históricos, por lo que estas entidades siempre contendrán únicamente los registros a la mensualidad correspondiente:

- "*Usuario – FacturesMensuals*": La entidad "*FacturesMensuals*" tiene los registros diferenciados por el número de bloque, y como sabemos que para cada bloque habrán 3 tipos de facturas (Gas, Agua y Luz) por mensualidad, podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 3 registros de la entidad "*FacturesMensuals*".

- "*Usuario – LecturesMensuals*": La entidad "*LecturesMensuals*" tiene los registros diferenciados por el número de bloque, por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 1 registro de la entidad "*LecturesMensuals*".

- "*Usuario – ConsumsGeneralsMensuals*": La entidad "*ConsumsGeneralsMensuals*" tiene los registros diferenciados por el número de bloque, por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 2 registro de la entidad "*ConsumsGeneralsMensuals*".

- "*Usuario – TotalsFacturesMensuals*": La entidad "*TotalsFacturesMensuals*" tiene los registros diferenciados por el número de bloque, y como sabemos que para cada bloque habrán 3 tipos de facturas (Gas, Agua y Luz) por mensualidad, podemos afirmar que cada 23…24 registros de la entidad "*Usuario*" están relacionados con 3 registros de la entidad "*FacturesMensuals*".

- *Entidades de registros temporales*

Las relaciones que se describirán a continuación, están basadas en unos registros temporales, los cuales se crean en el momento de la subida de datos por parte del administrador y que se almacenan en dichas entidades a la espera de que estén todos introducidos, entonces una vez se han subido todos los datos de la mensualidad correspondiente y se procede a realizar el cálculo de todos los costes, estos registros son eliminados, por lo que en la gran mayoría de ocasiones estas entidades se encontrarán vacías.

- "*Usuario – FacturesMensualsTemporals*": La entidad "*FacturesMensualsTemporals*" tiene los registros diferenciados por el número de bloque, y como sabemos que para cada bloque habrán 3 tipos de facturas (Gas, Agua y Luz) por mensualidad, podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 0…3 registros de la entidad "*FacturesMensualsTemporals*".

- "*Usuario – LecturesMensualsTemporals*": La entidad "*LecturesMensualsTemporals*" tiene los registros diferenciados por el número de bloque, por lo que podemos afirmar que cada 23...24 registros de la entidad

"*Usuario*" están relacionados con 0…1 registros de la entidad "*LecturesMensualsTemporals*".

- "***Usuario – CSVFiles***": La entidad "*CSVFiles*" tiene los registros diferenciados por el número de bloque, por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 0…1 registros de la entidad "*CSVFiles*".

- *Entidad de registros de valores estadísticos*

La relación que se describirá a continuación, está basada en los registros estadísticos históricos que tienen como finalidad permitir al usuario una comparativa diferentes consumos a lo largo del último año.
Hace falta añadir que, en el arranque de la aplicación, como esta se realizará con la subida de la primera mensualidad, ya se dispondrá de registros en la entidad.

- "***Usuario – Gráficos***": La entidad "*Gráficos*" tiene los registros diferenciados por número de bloque, por lo que podemos afirmar que cada 23...24 registros de la entidad "*Usuario*" están relacionados con 0..* registros de la entidad "*Gráficos*".

- *Entidad de registros de notificaciones*

La relación que se describirá a continuación, está basada en los registros de las notificaciones de la aplicación que tienen como finalidad informar al usuario de las subidas de cada mensualidad.
Hace falta añadir que, en el arranque de la aplicación, como este se realizará con la subida de la primera mensualidad, ya se enviará la primera notificación a cada usuario.

- "***Usuario – Mensaje***": La entidad "*Mensaje*" tiene los registros diferenciados por usuario, por lo que podemos afirmar que cada 1 registro de la entidad "*Usuario*" está relacionado con 1...* registros de la entidad "*Mensaje*".

- *Entidad de registros usuario*

La relación que se describirá a continuación, está basada en los registros de usuario relacionados con la entidad "*User*" del propio sistema, con la finalidad de poder aprovechar la funcionalidad de autenticación que nos proporciona el framework *Django*.

- "***Usuario – User***": La entidad "*Usuario*" tiene los registros diferenciados a través de la clave foránea user, ya que cada registro de la entidad "*Usuario*" está relacionado de 1 a 1 con la entidad "*User*".

- *Entidad de registros de mapeo*

La relación que se describirá a continuación, está basada en los registros de mapeo entre usuarios y pisos, que tiene como finalidad identificar a que *id_piso* corresponde cada usuario.
Hace falta añadir que, en el arranque de la aplicación, se llevará a cabo la subida del archivo de mapeo, por lo que ya tendremos relacionados los usuarios con su *id_piso*.

- "***Usuario – Mapping***": La entidad "*Mapping*" tiene los registros diferenciados por id_piso, por lo que podemos afirmar que cada 1 registro de la entidad "*Usuario*" está relacionado con 1 registro de la entidad "*Mapping*".

Desarrollo del proyecto

En este apartado se describirá toda la estructura del proyecto, mostrando las decisiones e ideas tomadas para el correcto desarrollo de la aplicación, junto con algunas porciones de código de las diferentes funcionalidades.

- *Estructura del proyecto*

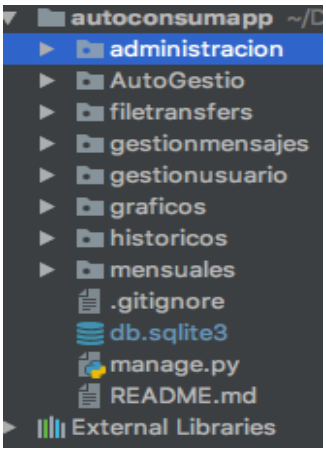
Para la creación de la estructura del proyecto se han utilizado las herramientas que facilita el propio *framework Django*.
Inicialmente se ha creado el proyecto general con el nombre *AutoGestio*, con el siguiente comando:

- `$ python manage.py startproject AutoGestio`

Una vez creado el proyecto general, se ha tomado la decisión de dividir el proyecto en diferentes aplicaciones, para de esta forma conseguir desacoplar las diferentes funcionalidades de la aplicación y conseguir también una mayor organización de las partes del código.
Para la creación de cada módulo o app, se ha utilizado el siguiente comando dentro de la carpeta del proyecto general:

- `$ python manage.py startapp ['nombre app']`

Quedando finalmente la estructura del proyecto, tal y como se muestra en la captura de pantalla, donde se ven claramente diferenciados los diferentes módulos de aplicación, junto con la carpeta *AutoGestio* que es donde se encuentran los archivos de configuraciones y el archivo urls.py general de la aplicación.
Se puede observar que también se ha creado la base de datos *db.sqlite3*, ya que se trata del entorno de test, en el caso del entorno de producción se realiza la misma creación, pero del tipo *PostgreSQL*:



- *Patrón de diseño utilizado*

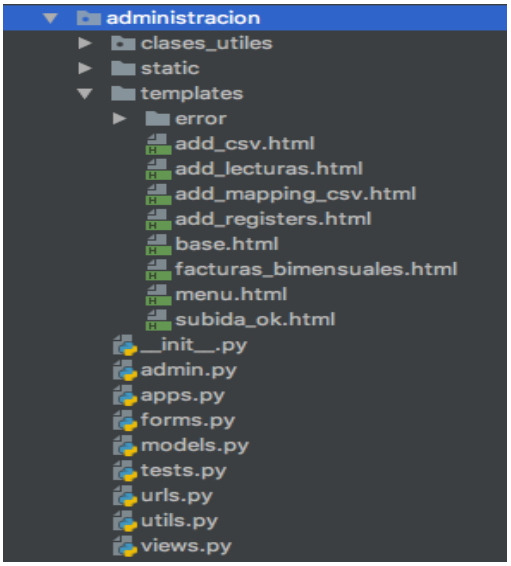
El propio *framework django*, está basado en el patrón de diseño *MTV(Modelo – Template – Vista)* el cual es prácticamente en una copia del patrón *MVC (Modelo – Vista - Controlador)* el cual se trata de un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.
Donde para lograrlo propone la construcción de 3 elementos *modelo, vista y controlador*, definiendo por un lado componentes para la representación de la información y por otro lado para la interacción del usuario.
Se dice que prácticamente son iguales ya que la función del elemento *Modelo* es la misma en ambos casos, por lo referente al elemento *Template*, del patrón *MTV*, este corresponde al elemento *Vista* del patrón *MVC* y por último el elemento *Vista* del patrón *MTV* actúa de igual forma que el elemento *Controlador* del patrón *MVC*, dónde:

- *Modelo*: Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto las consultas como actualizaciones, implementando también privilegios de acceso que se hayan descrito en la lógica de negocio.

- *Template*: Presenta el '*modelo*' (información y lógica de negocio) en un formato adecuado (*HTML*) para interactuar (*interfaz de usuario*), por tanto requiere de dicho '*modelo*' la información que debe representar como salida.

- *Vista*: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al '*modelo*' cuando se hace alguna solicitud sobre la información. Por lo que se podría decir que la vista hace de intermediaria entre el '*Template*' y el '*Modelo*'.

Estos elementos quedan correctamente definidos para cada módulo o app del proyecto, tal y como se muestra en la siguiente captura de pantalla:



- *Diseño estructura de las rutas*

El *framework Django*, utiliza como gestor de las rutas de la aplicación, un fichero llamado '*urls.py*' el cual se encarga de llevar todas las relaciones entre las rutas y sus vistas correspondientes. Este fichero se crea en el mismo momento que el proyecto general, almacenándose en la carpeta donde se encuentran también los ficheros de configuraciones. Por otra parte, dentro de cada módulo también se ha creado un fichero urls.py para permitir de esta forma una gestión de rutas autónoma de cada módulo.
Para poder permitir dicha gestión autónoma, primero es necesario definir el espacio de nombres que relaciona el gestor de rutas de cada módulo con el gestor general, tal y como se muestra en la siguiente captura de pantalla:

```
urlpatterns=[

#Admin
url(r'^admin/',admin.site.urls),

#Namespace's
url(r'^',include('administracion.urls',namespace='administracion')),
url(r'^',include('gestionusuario.urls',namespace='gestionusuario')),
url(r'^',include('historicos.urls',namespace='historicos')),
url(r'^',include('mensuales.urls',namespace='mensuales')),
url(r'^',include('graficos.urls',namespace='graficos')),
url(r'^',include('gestionmensajes.urls',namespace='gestionmensajes')),

#Pantallaprincipal
url(r'^main/$',MenuView.as_view(),name="main"),
```

Una vez definido el espacio de nombres de los diferentes gestores de rutas, ya se pueden definir las relaciones "*Ruta – Vista*" para cada módulo, como se muestra en el siguiente ejemplo:

```
urlpatterns                                          =                                          [

    url(r'^month/consum/1$',ConsumoMensual1View.as_view(),
        name="month_consum_bloc1"),
    url(r'^month/consum/2$',ConsumoMensual2View.as_view(),
        name="month_consum_bloc2"),
    url(r'^month/consum/3$',ConsumoMensual3View.as_view(),
        name="month_consum_bloc3"),
    url(r'^month/consum/4$',ConsumoMensual4View.as_view(),
        name="month_consum_bloc4"),
    url(r'^month/facturas/1$',FacturasMensuales1View.as_view(),
        name="month_facturas_bloc1"),
    url(r'^month/facturas/2$',FacturasMensuales2View.as_view(),
        name="month_facturas_bloc2"),
    url(r'^month/totales/1$',TotalsMensuals1View.as_view(),
        name="month_totales_bloc1"),
    url(r'^month/totales/2$',TotalsMensuals2View.as_view(),
        name="month_totales_bloc2"),

]
```

- *Estructura ficheros de configuración*

Tal como se ha mencionado anteriormente, el propio *framework Django*, en el momento de la creación del proyecto general, se encarga de preparar el archivo '*settings.py*' el cual contiene todos los parámetros de configuración del proyecto.
Entonces como la finalidad de dicha aplicación es estar en producción, pero con un constante mantenimiento en un entorno de test, para una mayor comodidad se ha decidido crear dos ficheros de configuración, uno para el entorno de test apuntando a la correspondiente base de datos '*sqlite3*' y otro para el entorno de producción apuntando a la base de datos real '*PostgreSQL'*.
Para realizar dicha modificación, es necesario modificar primero el fichero base del proyecto nombrado '*manage.py*', para indicarle que archivo de configuraciones debe servir según el entorno en el que se encuentre, tal y como se muestra a continuación:

```
if __name__=="__main__":
    ifENVIRONMENT=='TEST':
        os.environ.setdefault(
            "DJANGO_SETTINGS_MODULE",
            "AutoGestio.settings.test environment"
        )
    else:
        os.environ.setdefault(
            "DJANGO_SETTINGS_MODULE",
            "AutoGestio.settings.production environment"
        )
```

Donde la variable '*ENVIRONMENT*' es recibida por parte del servidor que sirve la aplicación.

Una vez configurado el fichero '*manage.py*', se ha creado el fichero '*test_environment.py*' para el entorno de test y el fichero '*production_environment.py*' para el entorno de producción, tal y como sigue:

En el fichero de configuración encontramos diferentes apartados, los cuales son necesarios de informar para el correcto funcionamiento de la aplicación.
Muchos de los apartados ya vienen correctamente configurados, por lo que no es necesario modificarlos, pero a continuación se describen los que requieren su modificación:

- '*INSTALLED_APPS*': Es necesario añadirle los módulos o aplicaciones que se van a utilizar en el proyecto:

```
INSTALLED_APPS=[
    'administracion',
    'gestionusuario',
    'historicos',
    'mensuales',
    'graficos',
    'gestionmensajes',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
   'django.contrib.staticfiles',
]
```

- '*TEMPLATES*': Es necesario definir donde se encontrará el directorio contenedor de los diferentes '*templates*', junto con los procesadores de contexto utilizados:

```
TEMPLATES=[
    {
        'BACKEND':'django.template.backends.django.DjangoTemplates',
        'DIRS':[os.path.join(BASE_DIR,'templates')],
        'APP_DIRS':True,
        'OPTIONS':{
            'context processors':[
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'administracion.clases utiles.processors.myprocessors',
            ],
        },
    },
]
```

- '*DATABASES*': Es necesario definir qué tipo de base de datos utilizará nuestro proyecto y también la localización en el proyecto (Base de datos para el entorno de TEST):

```
DATABASES={
    'default':{
        'ENGINE':'django.db.backends.sqlite3',
        'NAME':os.path.join(BASE_DIR,'../db.sqlite3'),
    }
}
```

- '*MEDIA*' y '*STATIC*': Es necesario definir la ruta y la localización de los ficheros estáticos y los ficheros media:

```
STATIC_URL='/static/'
STATIC_ROOT=os.path.join(BASE_DIR,'staticfiles')

BASE_DIR=os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

MEDIA_ROOT='media/'
MEDIA_URL = '/media/'
```

- '*Auth System Config.*': Es necesario definir las rutas por defecto de redirección para el inicio o cierre de sesión en el sistema de autenticación:

```
LOGIN_URL='/'
LOGIN_REDIRECT_URL='/main'
LOGOUT_REDIRECT_URL = '/'
```

- '*Email Settings*': Es necesario definir la configuración del servidor SMTP, ya que en mi caso lo utilizo para enviar el correo de recuperación de contraseña:

```
EMAIL_HOST="smtp.gmail.com"
EMAIL_PORT="587"
EMAIL_HOST_USER="xxxxxxxx@xxxxxx.com"
EMAIL_HOST_PASSWORD="xxxxxx"
EMAIL_USE_TLS = True
```

Desarrollo de las funcionalidades

En este apartado se intentarán plasmar las diferentes ideas tomadas para el desarrollo de las funcionalidades establecidas.

Módulo de Administración

Dicho módulo tiene como finalidad gestionar todas las funcionalidades del sistema encargadas de la administración del mismo, es decir, todas aquellas funcionalidades relacionadas con la subida de datos y su gestión.

Subida de datos

El principal requerimiento del cliente, para esta funcionalidad, consiste en llevar un control cronológico basado en la subida de datos, para no permitir ejecutar los cálculos mensuales sin previamente haber subido correctamente todos los datos en el sistema.
Para garantizar dicho requerimiento, he decidido crear una entidad temporal para cada tipo de datos a subir durante la mensualidad (archivos .csv, lecturas y facturas).
La finalidad de las entidades temporales no es otra que almacenar de forma temporal los datos correspondientes a la subida mensual, mientras todos ellos no se hayan subido en su totalidad.

De esta forma se pretende llevar un control durante toda la subida mensual para evitar la duplicidad de facturas, lecturas o archivos .csv, a través de diferentes consultas por parte de la clase implementada "*ExceptionsController*" hacia las entidades temporales.
La subida cronológica de los datos mensuales, también se garantiza mediante la clase "ExceptionsController", la cual establece el siguiente orden de subida:

- "*Archivo .csv -> Lecturas Mensuales -> Facturas Mensuales*"

Por último, este módulo también es el encargado de gestionar la subida del fichero .csv que contiene el mapeo entre los diferentes pisos de las comunidades de vecinos y sus id's de piso, para de esta forma lograr relacionar cada vecino con sus correspondientes valores estadísticos obtenidos a través de la subida de los ficheros .csv mensuales para cada bloque.

Gestión de Datos

Tal y como se ha comentado en el apartado anterior, la gestión de los datos no se puede realizar sin que previamente se haya subido toda la información necesaria para la mensualidad.
Entonces he decidido implementar unas varias funciones encargadas de comprobar, mediante varias consultas a través de la clase "*ExceptionsController*", si ya se encuentran todas las entidades temporales, necesarias para la gestión, disponibles.
Para tener en todo momento actualizado el resultado de dicha función en nuestro sistema, he decidido aprovechar los procesadores de contexto de Django, los cuales se encargan de *renderizar*, en todo momento, en el *Template* base de la aplicación el resultado de las funciones definidas en ellos, ya que dichos procesadores de contexto son siempre llamados antes de mostrar la página al usuario.
De esta forma al realizar cualquier subida de datos se comprueba si es la última necesaria para realizar la gestión y en caso afirmativo se muestra para el usuario administrador el apartado relacionado con el cálculo mensual.

- *Módulo de Gestión de Mensajes*

Dicho módulo es el encargado del envío masivo de mensajes, a cada uno de los vecinos, cuando se ha realizado la gestión de los datos mensuales, con la finalidad de informarles sobre la disponibilidad de la mensualidad.
El criterio empleado en este módulo, se basa en la obtención de una lista total de todos los usuarios registrados en el sistema, creando un mensaje vinculado a cada uno de ellos.

- *Módulo de Gestión de Usuarios*

Dicho módulo es el encargado de gestionar la autenticación del usuario en todos sus aspectos (iniciar sesión, cerrar sesión y crear nuevo usuario).
Esta gestión se realiza aprovechando el módulo de autenticación que proporciona el *framework Django*, pero con un añadido, el cual consiste en una entidad "*Usuario*" que extiende la entidad "*User*" del sistema *Django*, con la finalidad de permitir gestionar para cada usuario sus datos referentes a su piso.
De esta forma se logra la funcionalidad basada en el cambio de contraseña que se le proporciona a cada usuario en su apartado de preferencias.

- *Módulo de Gráficos*

Dicho módulo es el encargado de proporcionar a los usuarios una comparativa gráfica satisfaciendo los requerimientos funcionales 9, 10 y 11.
Esto se logra mediante accesos a la entidad "*Registros Graficos*", a través de la cual se obtiene la información requerida para cada uno de los requerimientos funcionales y se plasma en la interfaz gráfica con la ayuda de la librería "*Chart.js*".

- *Módulo de Históricos*

Dicho módulo es el encargado de satisfacer las consultas de los usuarios en el apartado Históricos de la aplicación.
Tales consultas son respuestas a través de *Query Sets* a las entidades de Históricos, con el filtro correspondiente a la consulta del usuario.

- *Módulo de Mensuales*

Dicho módulo es el encargado de proporcionar a los usuarios todos los datos correspondientes a la mensualidad en cuestión.
Para lograr obtener dichos datos, se han implementado las clases "*Escalera*" y "*Piso*", donde inicialmente se instancia la clase "*Escalera*" referente a una escalera en concreto, la cual se encarga de obtener todo el conjunto de pisos que pertenecen a dicha escalera a través de una consulta a la entidad "*Mapping*" y una vez obtenida la lista de pisos, se

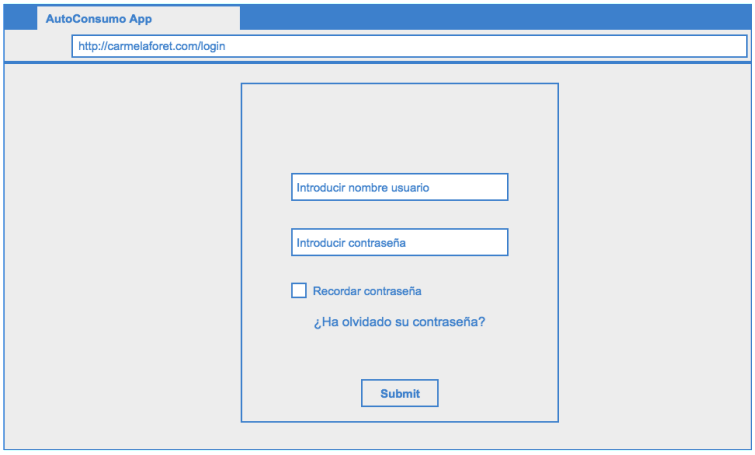
crea una instancia de la clase "*Piso*" para cada uno de estos, la cual se encarga de calcular los consumos mensuales para cada uno de ellos.
Por lo que se refiere a las facturas mensuales y los totales mensuales y de facturas, se obtienen mediante *Query Sets* a cada una de las entidades correspondientes.

Desarrollo de la interfaz gráfica

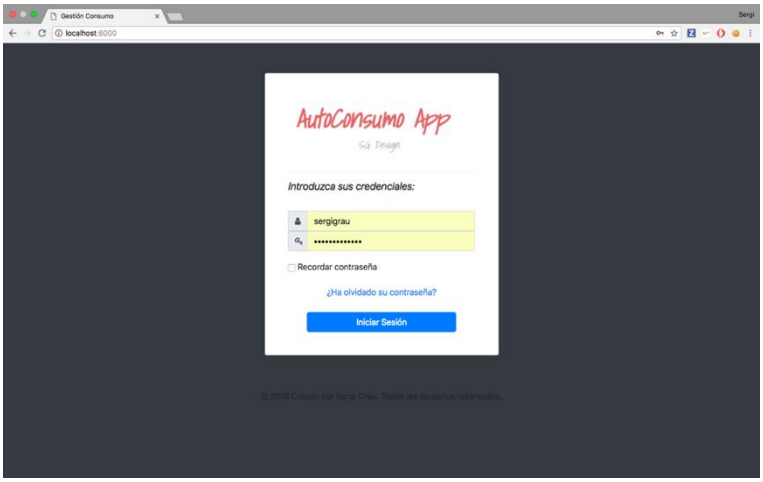
En este apartado se mostrará el resultado final de la interfaz gráfica, junto con los criterios seguidos para la realización de la misma.
El diseño de la interfaz gráfica se ha realizado mediante programación *Responsive* gracias a *Bootstrap*, para de esta forma lograr que la aplicación sea *multidispositivo*.
Durante todo este apartado, se realizará una comparativa del *mockup* [19] con la versión final de la interfaz.

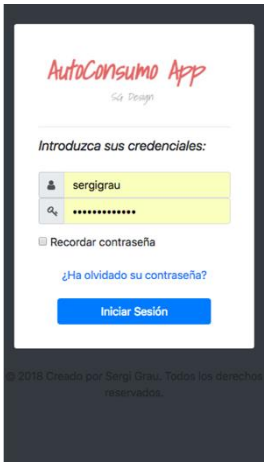
- *"Pantalla de inicio de sesión"*

  Diseño inicial:



  Diseño final para dispositivo web y móvil:

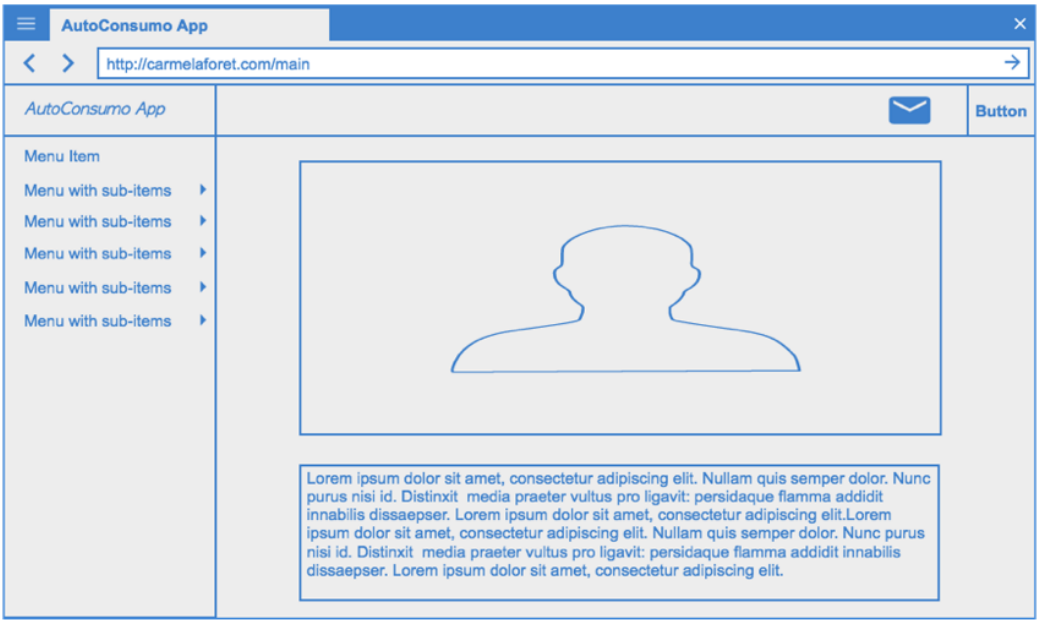




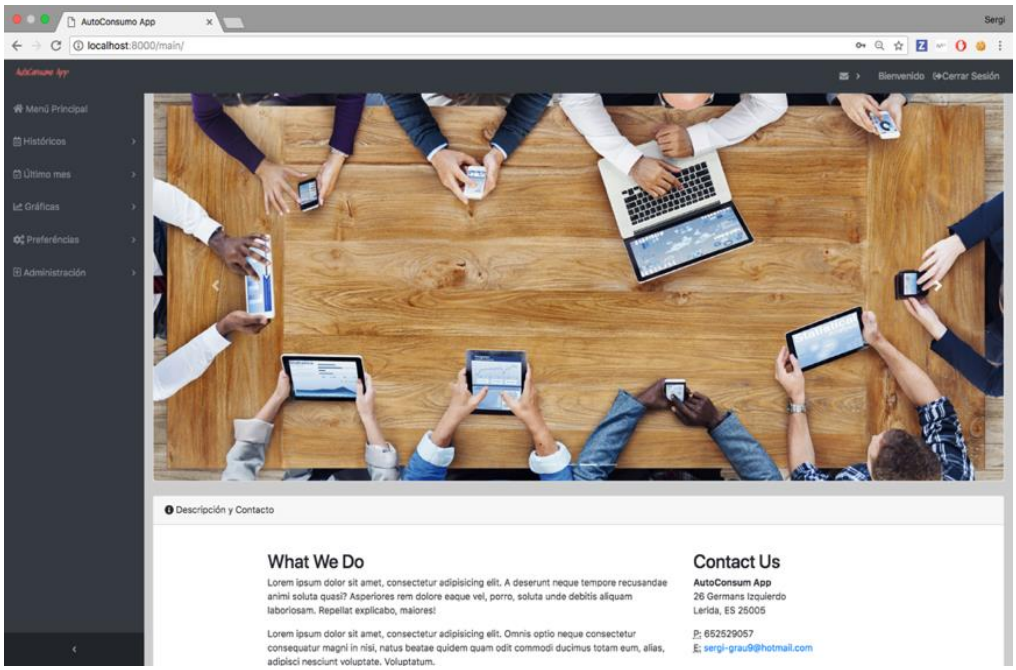
[19] Fotomontajes que permiten a los diseñadores gráficos y web mostrar al cliente como quedarán sus diseños.

- *“Pantalla menú principal”*

Diseño inicial:

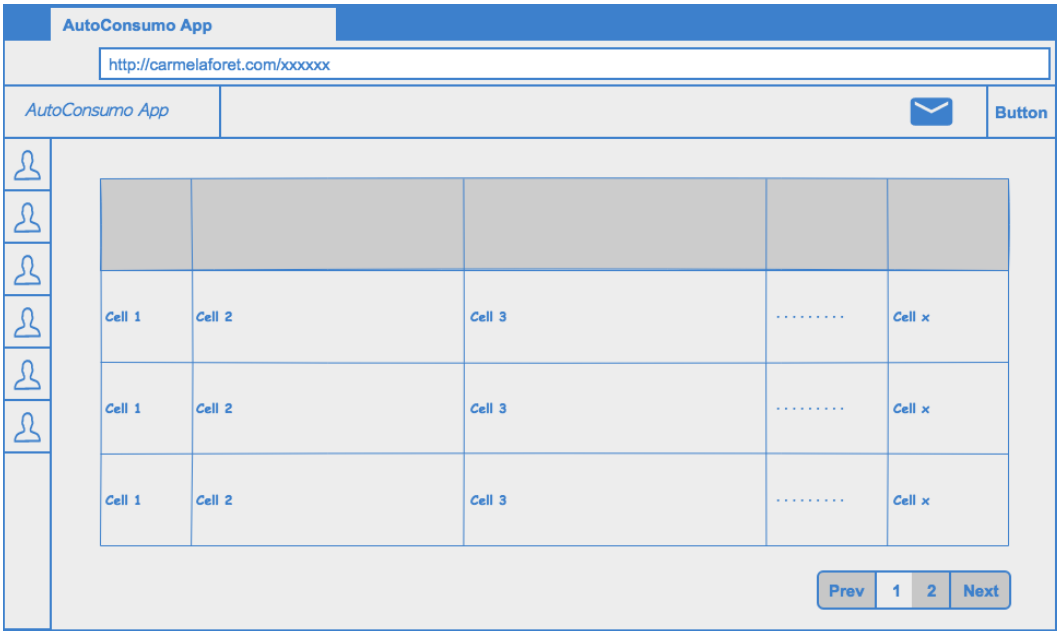


Diseño final para dispositivo web y móvil:

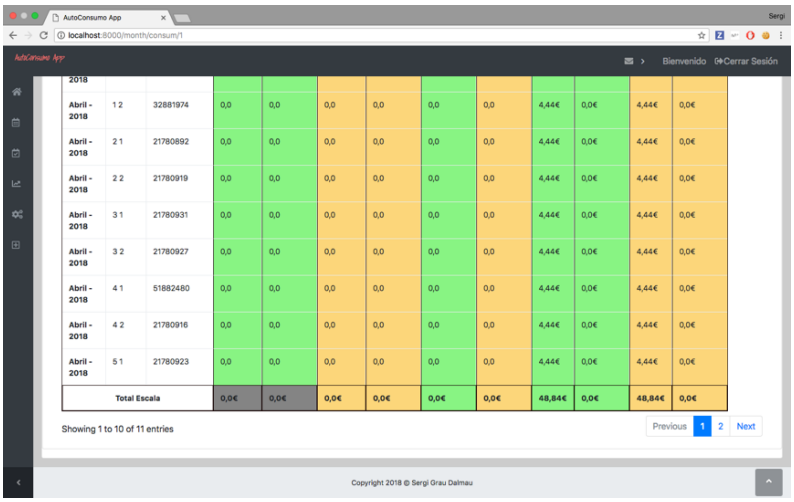


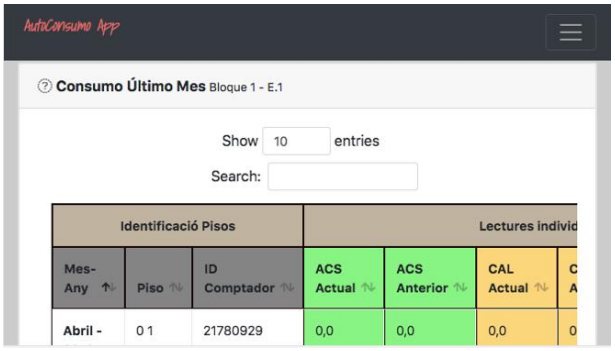
- *"Pantalla de consulta de datos en una única tabla"*

Diseño inicial:



Diseño final para dispositivo web y móvil:

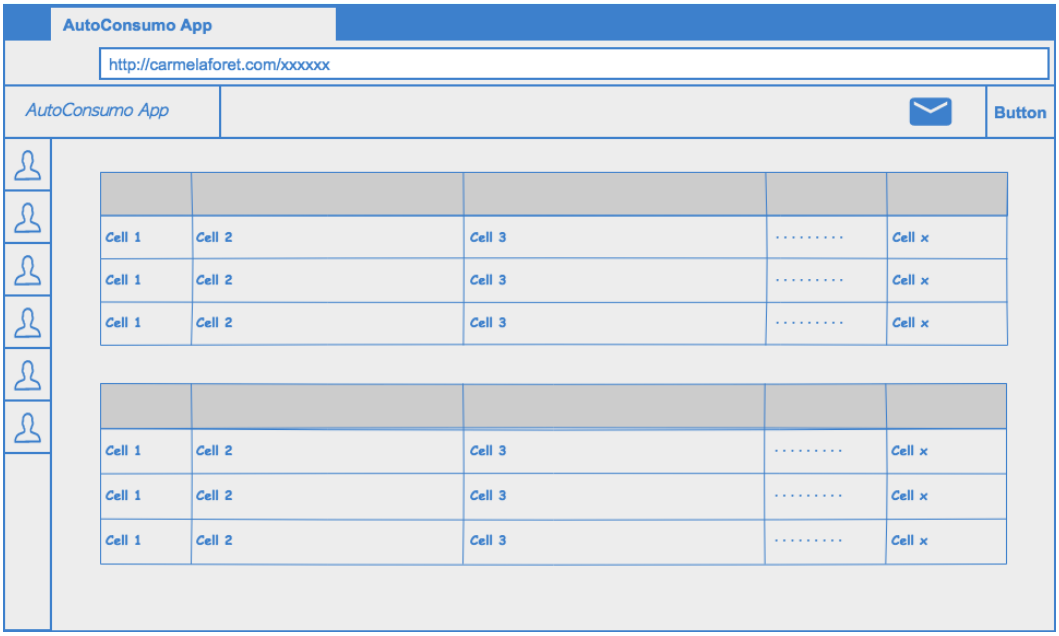




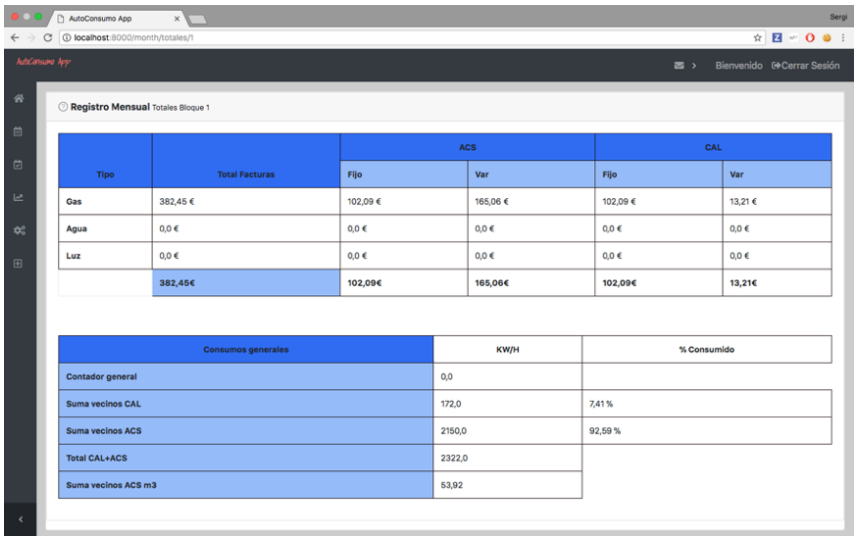
Gracias a la programación *Responsive* se proporciona al usuario un manejo de la tabla, de tal forma que mediante el táctil del dispositivo móvil le permite desplazarse por las diferentes columnas de la tabla sin que los otros elementos de la pantalla se muevan, de ahí el porqué de la imagen se ve cortada, ya que esta se puede consultar entera desplazándola en las diferentes direcciones mediante el sistema táctil.

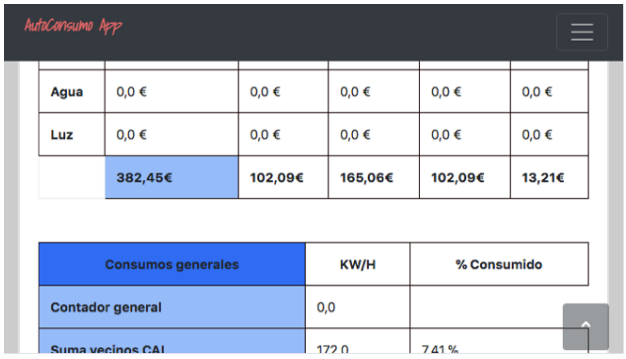
- *"Pantalla de consulta de datos en dos tablas"*

Diseño inicial:



Diseño final para dispositivo web y móvil:

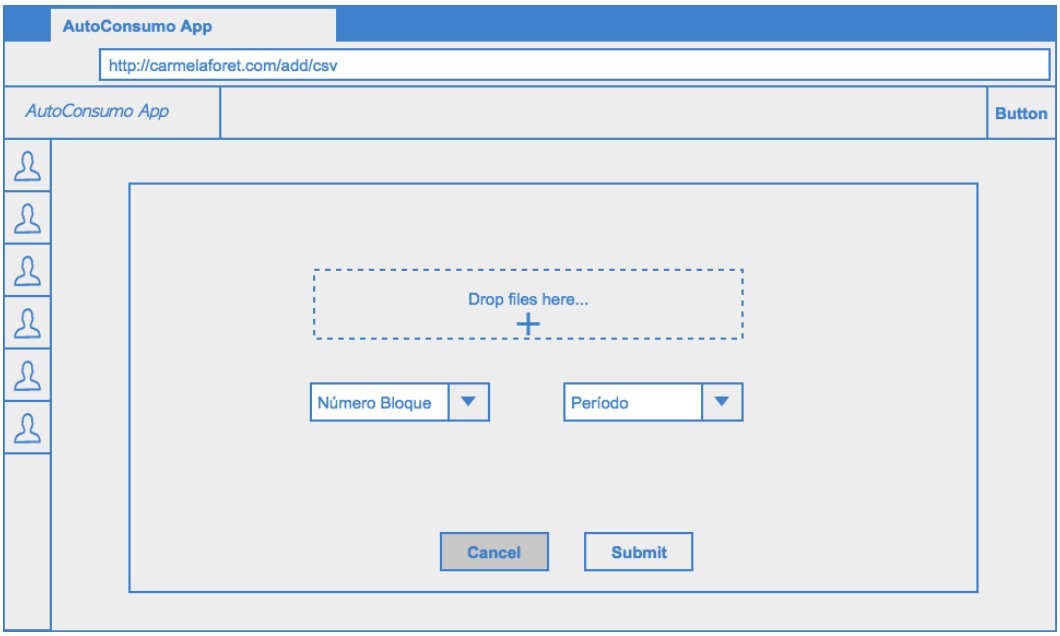




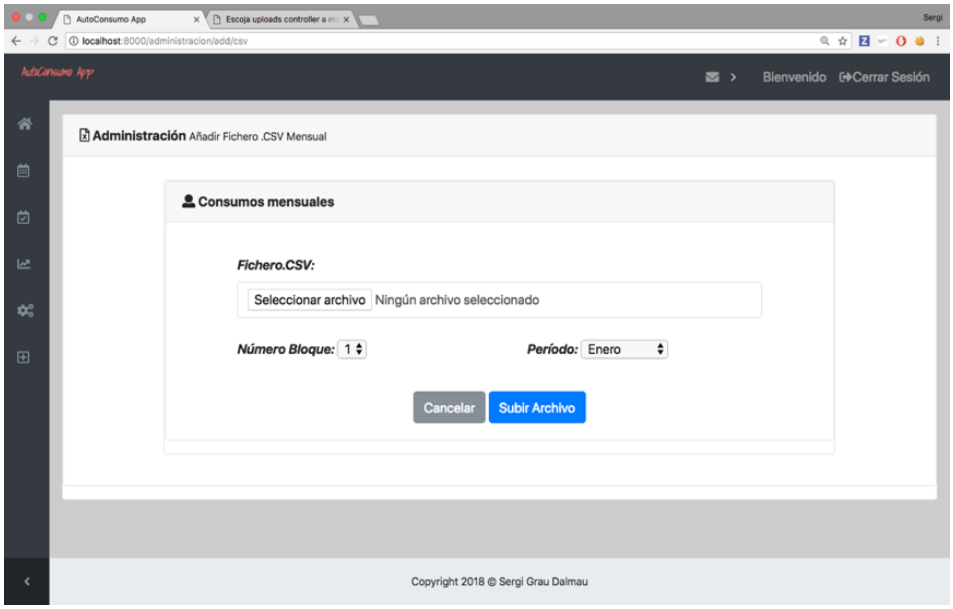
- *"Pantallas administración de datos"*

"Añadir fichero .csv estadístico"

Diseño inicial:

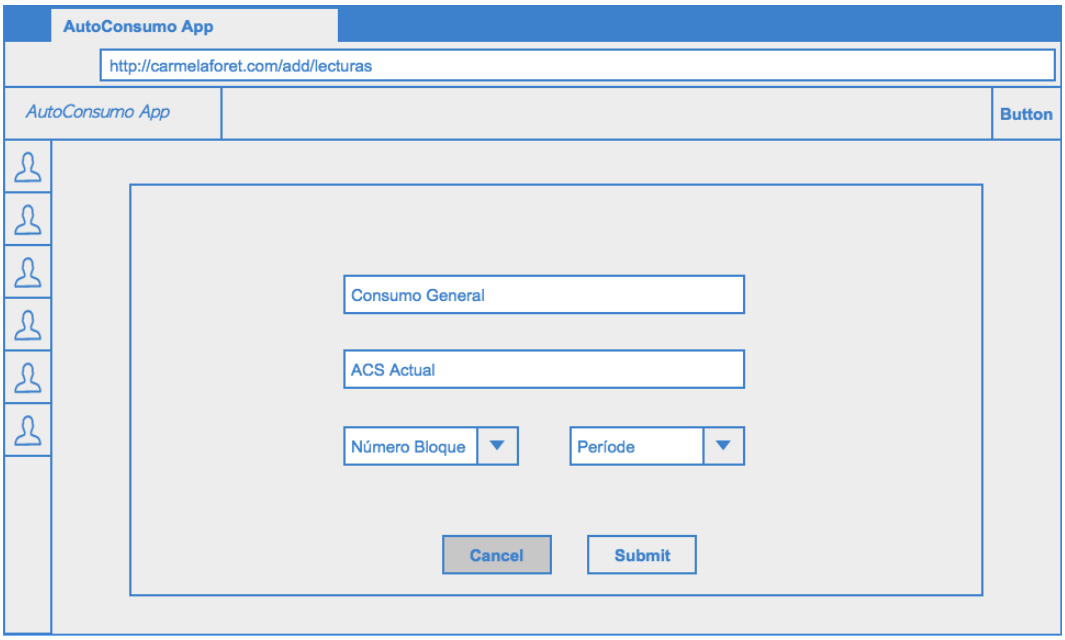


Diseño final para dispositivo web y móvil:

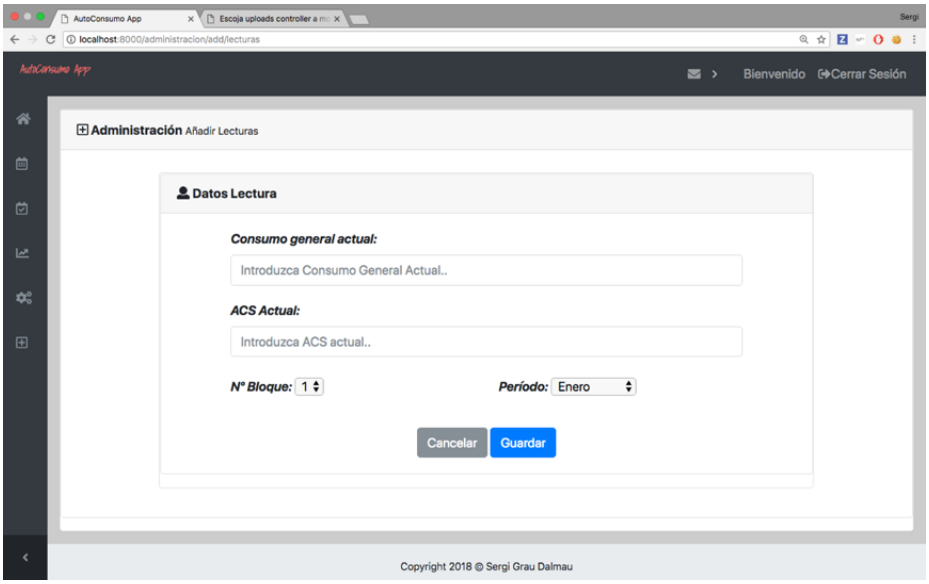


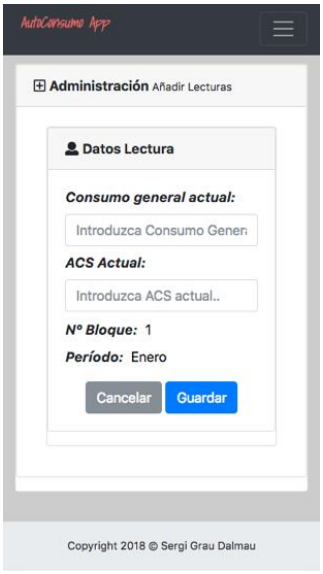
"Añadir lecturas mensuales"

Diseño inicial:



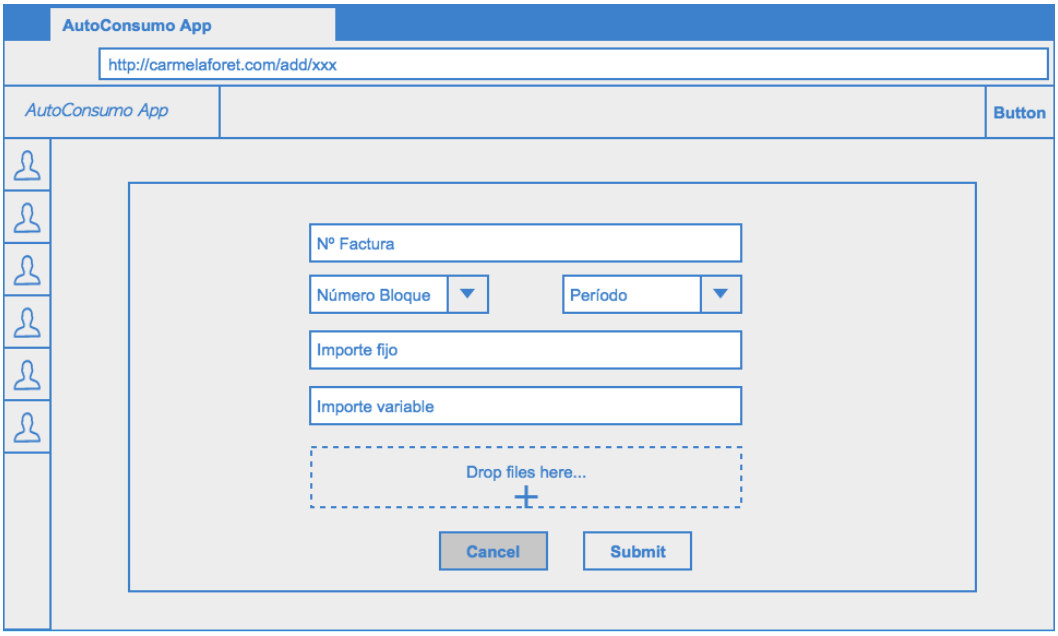
Diseño final para dispositivo web y móvil:



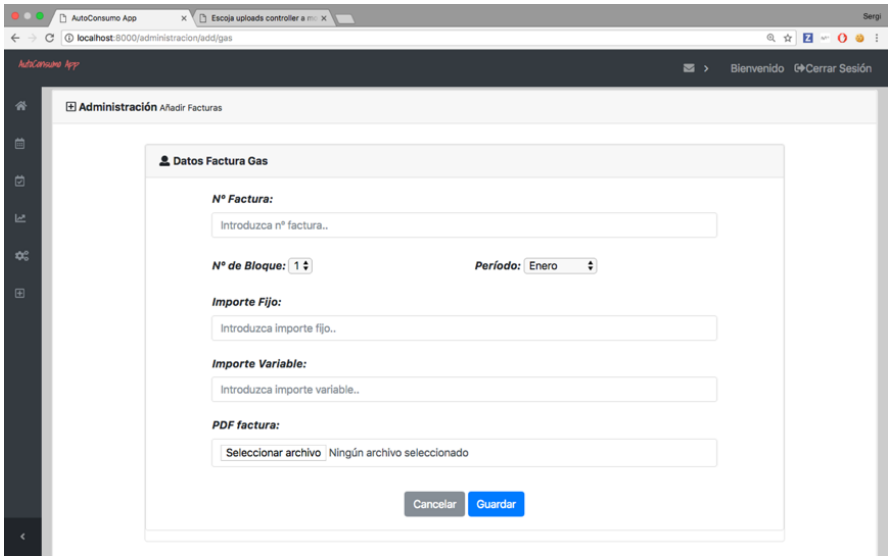


"Añadir facturas mensuales"

Diseño inicial:

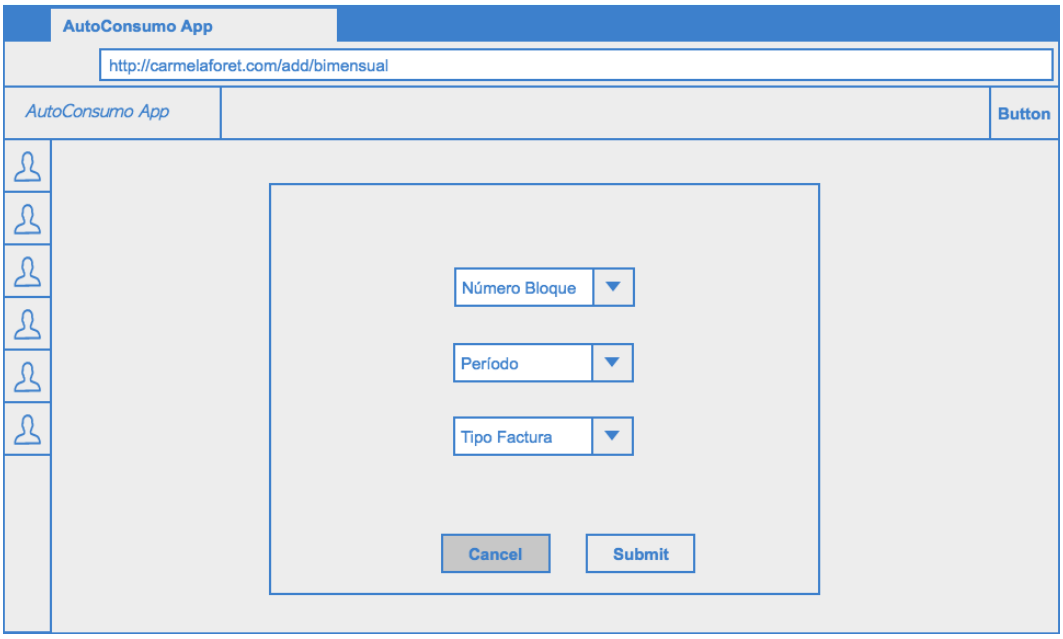


Diseño final para dispositivo web y móvil:

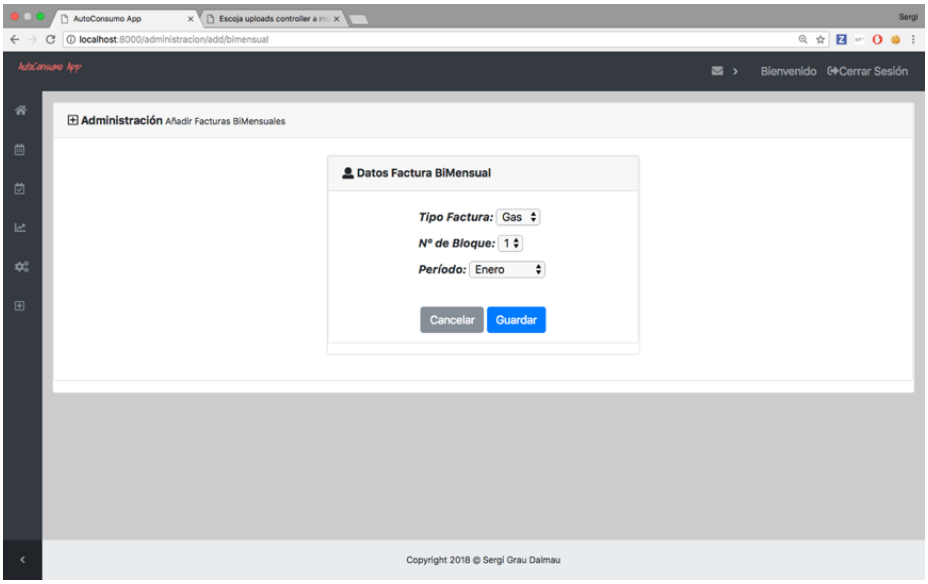


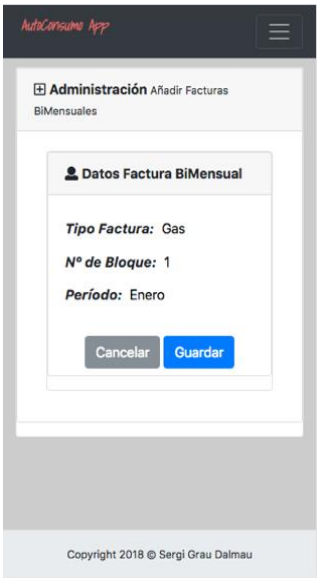
"Añadir facturas bimensuales"

Diseño inicial:



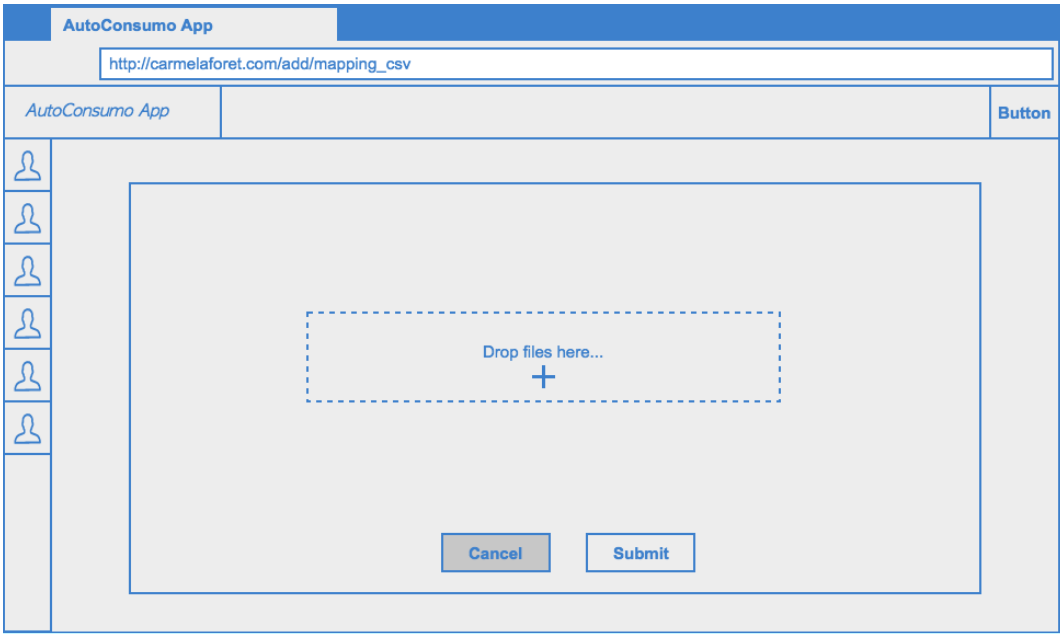
Diseño final para dispositivo web y móvil:



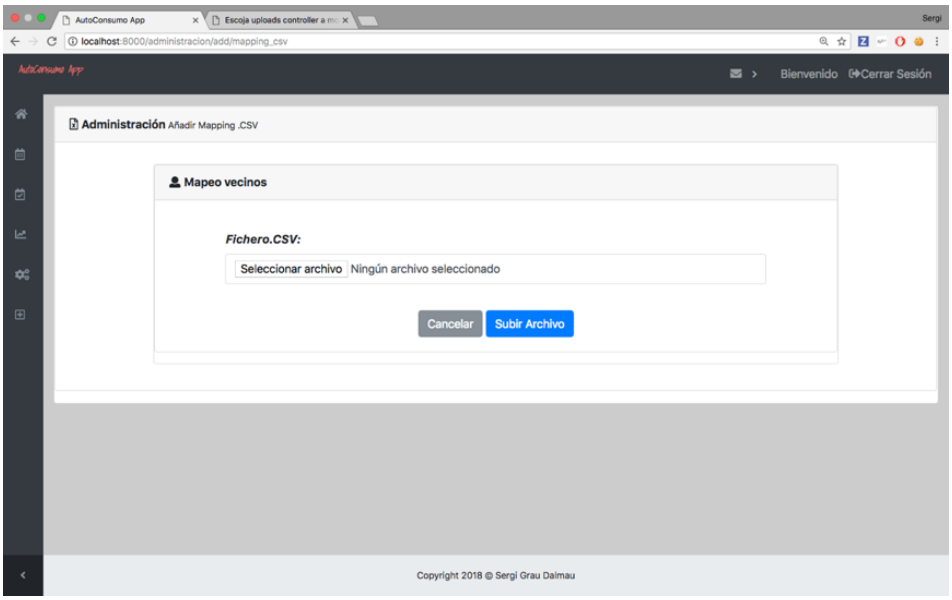


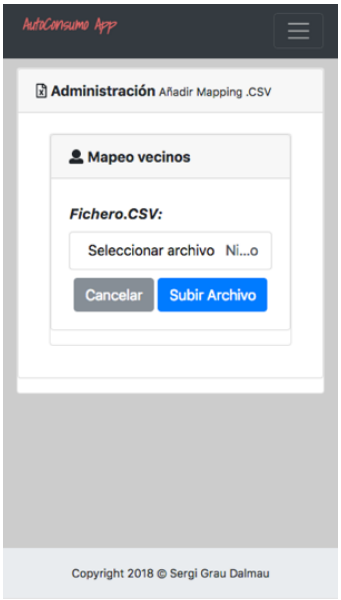
"Añadir fichero .csv de mapping"

Diseño inicial:



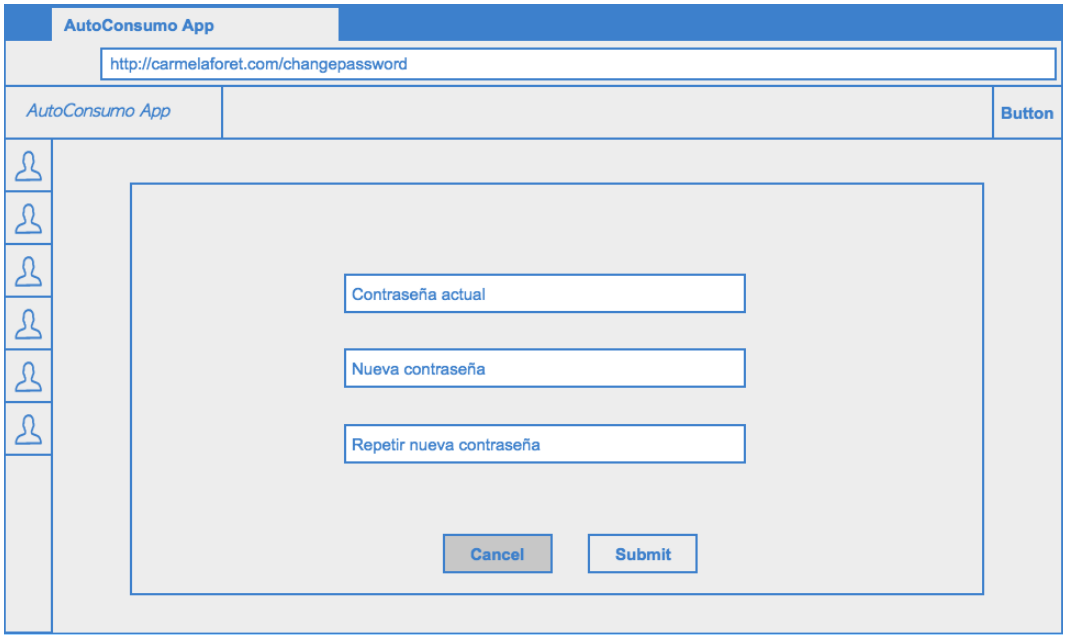
Diseño final para dispositivo web y móvil:



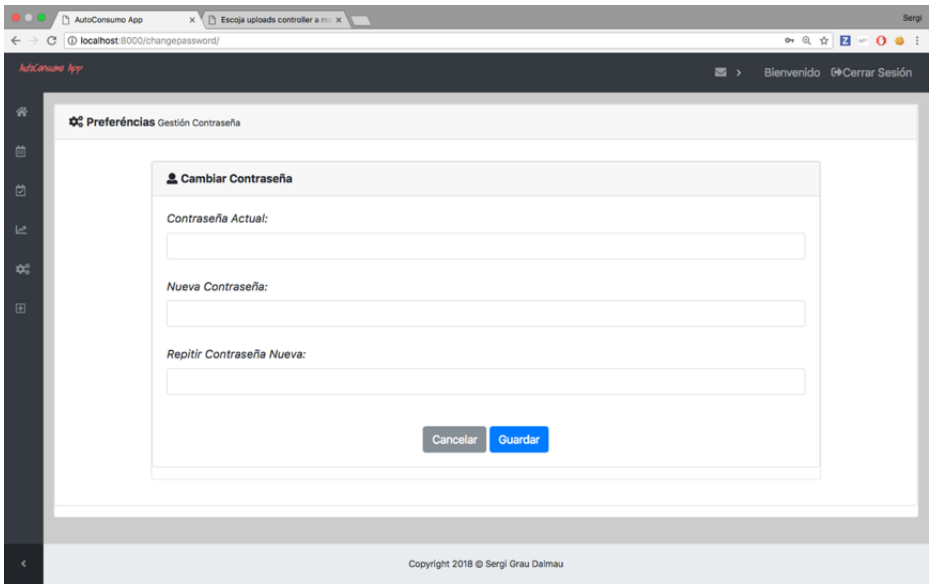


- *"Pantalla gestión contraseña usuario"*

Diseño inicial:



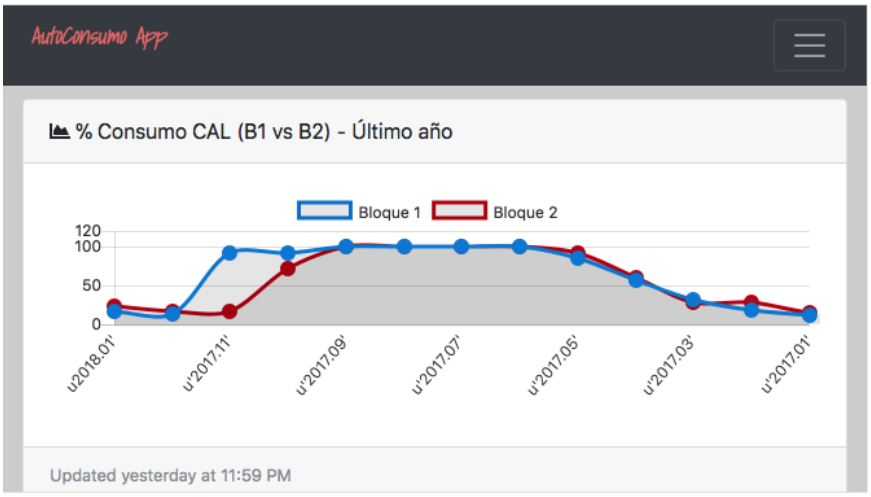
Diseño final para dispositivo web y móvil:



- *"Pantalla % Consumo CAL B1 vs B2"*

No se presenta el *Mockup* para esta pantalla, debido a que se trata del mismo que para las pantallas de tablas, con la diferencia que en vez de aparecer una tabla aparece una gráfica.

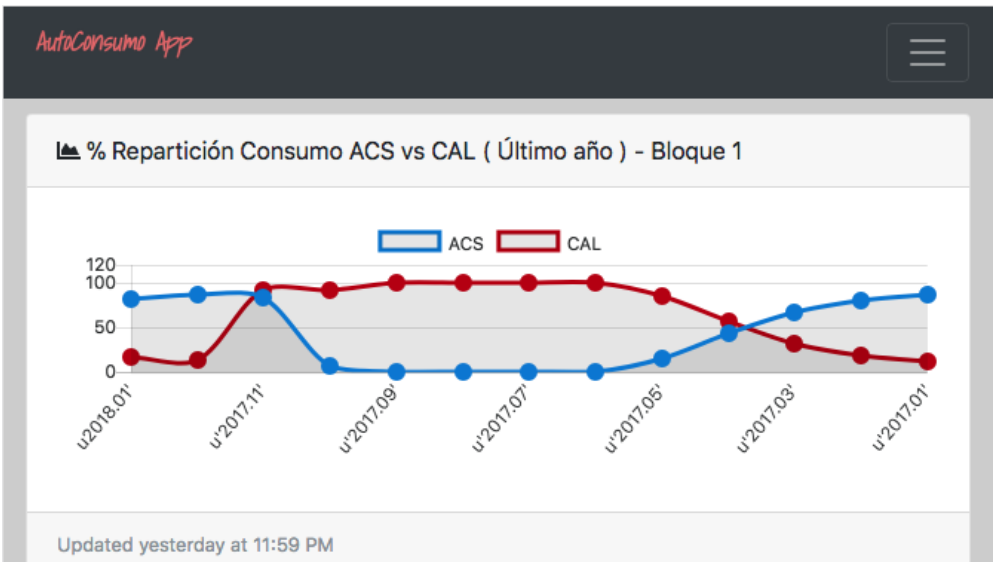
Diseño final para dispositivo web y móvil:



- *"Pantalla % Consumo ACS B1 vs B2"*

No se presenta el *Mockup* para esta pantalla, debido a que se trata del mismo que para las pantallas de tablas, con la diferencia que en vez de aparecer una tabla aparece una gráfica.

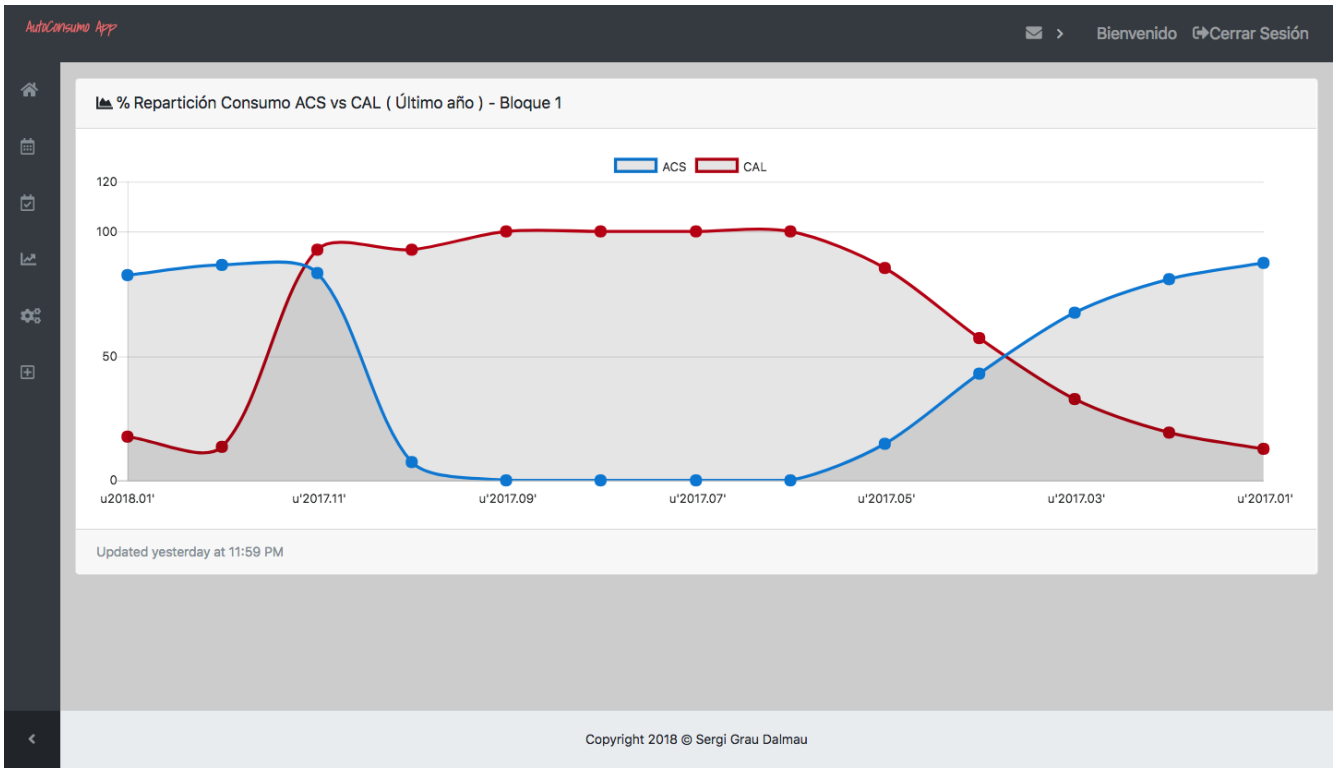
Diseño final para dispositivo web y móvil:

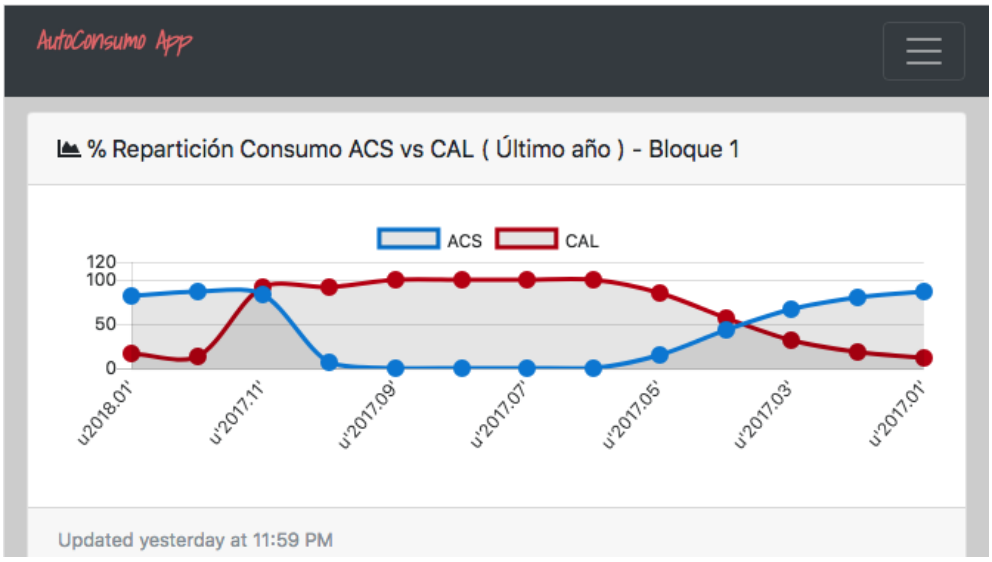


- *"Pantalla % Consumo ACS vs Consumo CAL Bloque x"*

No se presenta el *Mockup* para esta pantalla, debido a que se trata del mismo que para las pantallas de tablas, con la diferencia que en vez de aparecer una tabla aparece una gráfica.

Diseño final para dispositivo web y móvil:





Capítulo 5: Despliegue a producción

Para el despliegue de la aplicación al entorno de producción, se ha contratado un servidor VPS a través de la compañía *Hostinet*.

Dicho VPS dispone del sistema operativo *CentOs 7 minimal*, con privilegios root y sin administración.

A continuación, se detallarán todos los pasos seguidos para la correcta configuración del sistema operativo, para permitir que este aloje una aplicación Django trabajando con un motor de base de datos *PostgreSQL*.

Para ello es necesario disponer del servidor Apache, de un servidor ftp, de Python 2.7 (ya que se ha utilizado esta versión por temas de compatibilidad con el motor de base de datos) y otros complementos para abastecer los anteriormente nombrados.

Instalación y configuración servidor Apache

A continuación, se dará el listado de comandos necesarios para la correcta instalación y configuración del servidor Apache.
Antes de empezar con la instalación siempre es aconsejable actualizar el sistema operativo con su versión más reciente:

- `# sudo yum update -y`

Es necesario instalar el repositorio EPEL en el servidor:

- `# sudo yum install epel-release -y`

A continuación, se instala el software nano para el manejo de los archivos de configuración, el servidor Apache junto con el módulo wsgi, necesario para servir mediante Apache una aplicación Django:

- `# yum install nano`
- `# yum install httpd`
- `# yum install mod_wsgi`

Ahora toca encender el servidor y activarlo para su arranque automático al arrancar el VPS:

- `# systemctl start httpd.service`
- `# systemctl enable httpd.service`

Configuramos el firewall de centOs, para evitar bloqueos de este hacía apache:

- `# firewall-cmd --permanent --zone=public --add-service=http`
- `# firewall-cmd --permanent --zone=public --add-service=https`
- `# firewall-cmd --reload`
- `# systemctl restart httpd.service`

Una vez tenemos el servidor apache en correcto funcionamiento, es necesario configurar en él los parámetros necesarios para que este sepa que aplicación tiene que servir y como lo tiene que hacer.

Inicialmente accedemos al archivo base de configuración de apache:

- `# nano /etc/httpf/conf/httpd.conf`

Una vez tenemos el archivo de configuración abierto, debemos buscar y dejar como prosigue las siguientes líneas:

- *ServerName www.xxxxxxxxx.com:80 (Donde 'xxxxxxxxx' hace referencia al dominio y ':80' el puerto a través de el que nuestro servidor escuchará las peticiones (http) externas).*
- *ErrorDocument 500 /directorio/aplicacion/templates/error500.html (Para indicar el template que se mostrará en caso de obtener un error 500 en el servidor).*
- *ErrorDocument 404 /directorio/aplicacion/templates/error404.html (Para indicar el template que se mostrará en caso de obtener un error 404 en el servidor).*
- *ErrorDocument 402 /directorio/aplicacion/templates/error402.html (Para indicar el template que se mostrará en caso de obtener un error 402 en el servidor).*
- *IncludeOptional conf.d/*.conf (Es muy importante comprobar que esta línea este descomentada)*

Seguidamente, para tener una mejor organización del servidor, se va a crear un archivo de configuración único para nuestra aplicación:

- *# nano /etc/httpd/conf.d/django.conf*

En el cual vamos a crear el siguiente 'Virtual Host':

```
<VirtualHost *:80>
        Alias                                                   /static
/opt/djangoprojects/autoconsumapp/AutoGestio/staticfiles
        <Directory
/opt/djangoprojects/autoconsumapp/AutoGestio/staticfiles>
                Require all granted
        </Directory>
        Alias                                                 /static/admin
/opt/djangoprojects/autoconsumapp/AutoGestio/staticfiles/admin
        <Directory
/opt/djangoprojects/autoconsumapp/AutoGestio/staticfiles/admin>
                Require all granted
        </Directory>
        Alias /media /usr/share/httpd/media
        <Directory /usr/share/httpd/media>
                Require all granted
        </Directory>
        <Directory /opt/djangoprojects/autoconsumapp/AutoGestio>
                <Files wsgi.py>
                        Require all granted
                </Files>
        </Directory>
    WSGIScriptAlias                                                 /
/opt/djangoprojects/autoconsumapp/AutoGestio/wsgi.py
```

```
    WSGIDaemonProcess                AutoGestio                python-
    path=/opt/djangoprojects/autoconsumapp:/opt/djangoprojects/tfgpytho
    n27/lib64/python2.7/site-packages
         WSGIProcessGroup AutoGestio
</VirtualHost>
```

En este archivo como podemos comprobar le hemos indicado a nuestro servidor:

- Línea 1 a 4, donde y como se tiene que servir de los archivos estáticos de la aplicación.
- Línea 5 a 8, donde y como se tiene que servir de los archivos estáticos para la plataforma de administración de la aplicación.
- Línea 9 a 12, donde y como se tiene que servir de los archivos media de la aplicación.
- Línea 13 a 18, donde encontrar el fichero wsgi.py de la aplicación Django.
- Línea 19 a 21, donde encontrar el entorno Python para poder servir la aplicación Django.
- Línea 22, se ha asignado la ejecución de la aplicación en el grupo 'AutoGestio'.

Instalación y configuración servidor ftp

A continuación, se dará el listado de comandos necesarios para la correcta instalación y configuración del servidor ftp, el cual nos va a servir para una correcta transferencia de archivos con el servidor VPS.

Procedemos a realizar la instalación del servidor:

- `# yum install vsftpd`

Lo arrancamos y lo activamos para su automático arranque al iniciar el servidor VPS:

- `# systemctl start vsftpd.service`
- `# systemctl enable vsftpd.service`

Para una mayor seguridad del servidor, desactivamos su acceso anónimo, a través de su archivo de configuración que comúnmente se encuentra en '/etc/vsftpd/vsftpd.conf' y cambiamos la siguiente línea del archivo tal y como prosigue:

- `anonymous_enable=NO`

Restringimos el acceso a la carpeta '/home' cambiando la siguiente línea del archivo de configuración:

- `chroot_local_user=YES`

Habilitamos el modo pasivo del servidor, para evitar bloqueos del firewall o configuraciones NAT, cambiando las siguientes líneas del archivo de configuración:

- `allow_writeable_chroot=YES`
- `pasv_enable=Yes`
- `pasv_min_port=40000`
- `pasv_max_port=40100`

- `# setsebool -P ftpd_use_passive_mode=1`

Configuramos el sistema de seguridad SeLinux de CentOs, para evitar que bloquee el servidor ftp:

- `# sudo setsebool -P ftp_home_dir=1`

Creamos los usuarios, que van a acceder al servidor ftp:

- `# useradd -m ['username'] -s ['ruta directorio usuario']`
- `# passwd ['username']`
- `# ['Contraseña usuario']`

Creamos las reglas en el firewall de centOs:

- `# firewall-cmd --permanent --add-service=ftp`
- `# firewall-cmd --permanent --zone=public --add-port=20/tcp`
- `# firewall-cmd --permanent --zone=public --add-port=40000-40100`
- `# firewall-cmd -reload`
- `# systemctl restart httpd.service`

Damos permiso a los usuarios locales para acceder al resto de sistema de archivos:

- `# setsebool -P allow_ftpd_full_access 1`

Instalación y configuración de Python 2.7

A continuación, se dará el listado de comandos necesarios para la correcta instalación y configuración de Python.

Inicialmente es necesario instalar el paquete GCC:

- `# yum install gcc openssl-devel bzip2-devel`

Instalamos Python:

- `# yum install wget`
- `# cd /usr/src`
- `# wget https://www.python.org/ftp/python/2.7.14/Python-2.7.14.tgz`
- `# tar xzf Python-2.7.14.tgz`
- `# cd Python-2.7.14`
- `# ./configure --enable-optimizations`
- `# make altinstall`

A continuación, aislar el sistema referente a características necesarias para la aplicación en concreto, creamos un entorno virtual:

- `# yum -y install Python-pip`
- `# pip install virtualenv`

- *# cd /opt/djangoprojects*
- *# virtualenv tfgpython27*

Una vez tenemos el entorno virtual creado, le instalamos todos los requerimientos necesarios para la aplicación:

- *# source /opt/djangoprojects/tfgpython27/bin/actívate*
- *(tfgpython27)# pip install Django==1.11*
- *(tfgpython27)# pip install django-braces*
- *(tfgpython27)# pip install django-crispy-forms*
- *(tfgpython27)# deactivate*

Instalación y configuración motor PostgreSQL

A continuación, se dará el listado de comandos necesarios para la correcta instalación y configuración del motor de base de datos PostgreSQL.

Procedemos a instalar el servicio junto con todos sus complementos necesarios para el correcto funcionamiento:

- *# yum install postgresql-server postgresql-devel postgresql-contrib*

Una vez tenemos el servicio instalado, hemos de inicializarlo y arrancarlo:

- *# postgresql-setup initdb*
- *# systemctl start postgresql*

A continuación, vamos a acceder a uno de sus ficheros de configuración y a establecer los parámetros necesarios:

- *# nano /var/lib/pgsql/data/pg_hba.conf*

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
#host    all             all             127.0.0.1/32            ident
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
#host    all             all             ::1/128                 ident
host    all             all             ::1/128                 md5
```

Dejando los campos en rojo tal y como se muestran anteriormente, guardamos, configuramos el servicio para que se arranque automáticamente con el sistema y lo reiniciamos:

- *# systemctl restart postgresql*
- *# systemctl enable postgresql*

Configuramos el firewall de CentOs 7 para el servicio postgres y reiniciamos el servicio para que los cambios surjan efecto:

- *# firewall-cmd --permanent --zone=public --add-service=postgresql*
- *# firewall-cmd --permament --zone=public --add-port=5432/tcp*
- *# systemctl restart firewalld.service*

Una vez hecho esto, procedemos a crear la base de datos que utilizaremos en la aplicación y la asignaremos al usuario postgres que nos proporciona el mismo servicio, configurándola para el correcto funcionamiento con la aplicación Django:

- *# su - postgres*
- *psql*
- *postgres=> CREATE DATABASE autoconsum_db;*
- *postgres=> ALTER ROLE postgres SET client_encoding TO 'utf8';*
- *postgres=> ALTER ROLE postgres SET default_transaction_isolation TO 'read committed';*
- *postgres=> ALTER ROLE postgres SET timezone TO 'UTC';*
- *postgres=> GRANT ALL PRIVILEGES ON DATABASE autoconsum_db TO postgres;*
- *postgres=> \q*

Por último, para tener el motor de base de datos en correcto funcionamiento y permitir que nuestro servidor pueda escribir en la base de datos, hemos de indicarle al entorno SELinux la existencia de este, de la siguiente forma:

- *# setsebool -P httpd_can_network_connect_db on*

Una vez ya está el motor de base de datos totalmente configurado, queda indicarle a la aplicación Django, a través de su archivo de configuración, con qué base de datos tiene que trabajar:

```
DATABASES={
    'default':{
        'ENGINE':'django.db.backends.postgresql_psycopg2',
        'NAME':'autoconsum_db',
        'USER':'postgres',
        'PASSWORD':'xxxxxxxxx',
        'HOST':'h661.mi-vps.es',
        'PORT':'',
    }
}
```

Bibliografía

[1] **Django**. Django ORM. https://es.scribd.com/document/63901225/Django-ORM, 2011. *[Online, accedido el 10/03/2018]*

[2] **PostgreSQL**. PostgreSQL. https://es.wikipedia.org/wiki/PostgreSQL, 2018. *[Online, accedido el 10/03/2018]*

[3] **Apache**. Servidor HTTP Apache. https://es.wikipedia.org/wiki/Servidor_HTTP_Apache. *[Online, accedido el 10/03/2018]*

[4] **Python**. Python. https://es.wikipedia.org/wiki/Python, 2018. *[Online, accedido el 10/03/2018]*

[5] **VSFTPD**. Virtual Server FTPD. https://en.wikipedia.org/wiki/Vsftpd, 2018. *[Online, accedido el 10/03/2018]*

[6] **CentOs**. SO CentOs. https://es.wikipedia.org/wiki/CentOS, 2018. *[Online, accedido el 10/03/2018]*

[7] **HTML**. HTML 5. https://developer.mozilla.org/es/docs/HTML/HTML5, 2017. *[Online, accedido el 10/03/2018]*

[8] **Javascript**. Javascript. https://es.wikipedia.org/wiki/JavaScript, 2018. *[Online, accedido el 10/03/2018]*

[9] **Bootstrap 4**. Bootstrap 4. https://www.arweb.com/chucherias/%C2%BFque-es-bootstrap-y-como-funciona-en-el-diseno-web/ , 2014. *[Online, accedido el 10/03/2018]*

[10] **CSS**. Hojas de estilo en cascada. https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada , 2018. *[Online, accedido el 13/03/2018]*

[11] **jQuery**. jQuery. https://es.wikipedia.org/wiki/JQuery, 2018. *[Online, accedido el 13/03/2018]*

[12] **Popper JS**. Popper JS. https://programacion.net/articulo/10_librerias_de_css_y_javascript_que_es_increible_que_aun_no_conozcas_1410 , 2016. *[Online, accedido el 13/03/2018]*

[13] **Font awesome**. Font awesome. https://www.aquihaydominios.com/blog/font-awesom-que-es-y-como-se-usa/ , 2014. *[Online, accedido el 13/03/2018]*

[14] **Chart JS**. Chart JS. https://code.tutsplus.com/es/tutorials/getting-started-with-chartjs-introduction--cms-28278 , 2017. *[Online, accedido el 13/03/2018]*

[15] **ATOM**. Atom (Editor de textos). https://code.tutsplus.com/es/tutorials/getting-started-with-chartjs-introduction--cms-28278, 2018. *[Online, accedido el 13/03/2018]*

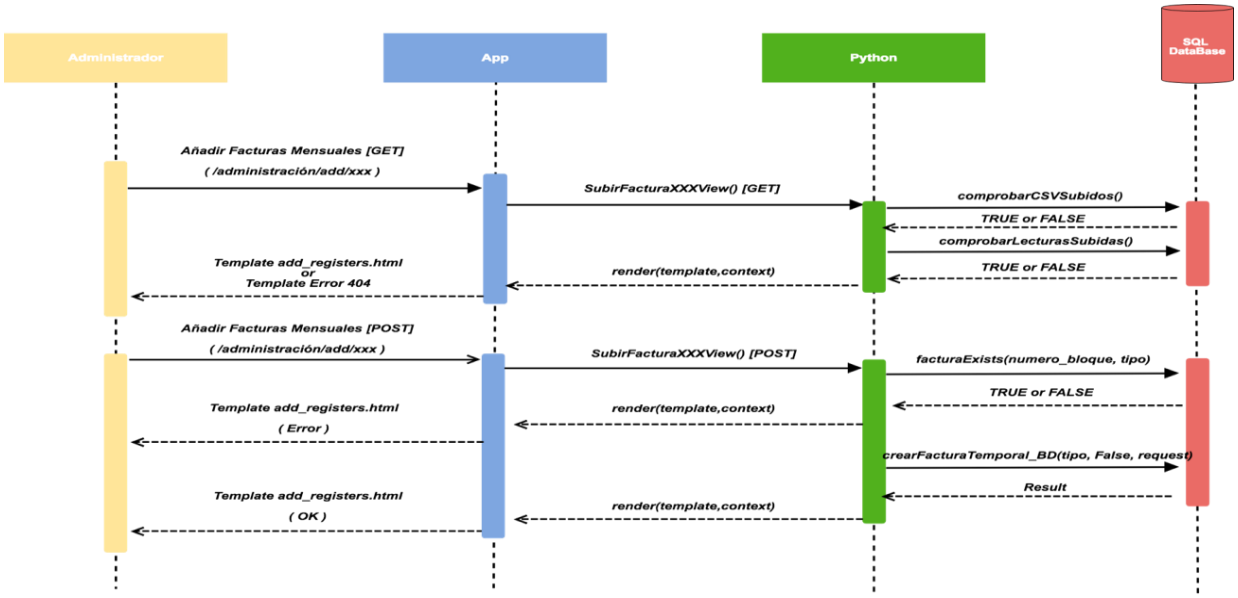
[16] **PyCharm**. PyCharm. https://en.wikipedia.org/wiki/PyCharm, 2018. *[Online, accedido el 13/03/2018]*

[17] **Cacoo**. Cacoo. http://aprendeenlinea.udea.edu.co/boa/contenidos.php/c7149543e9ad3ef665662a0aee87caf9/1035/estilo/aHR0cDovL2FwcmVuZGVlbmxpbmVhLnVkZWEuZWR1LmNvL2VzdGlsb3MvYXp1bF9jb3Jwb3JhdGl2by5jc3M=/1/contenido/1Que_es_Cacoo/cacoo.html , 2017. *[Online accedido el 13/03/2018]*

[18] **Pencil**. Pencil. https://pencil.evolus.vn/ , 2018. *[Online, accedido el 13/03/2018]*

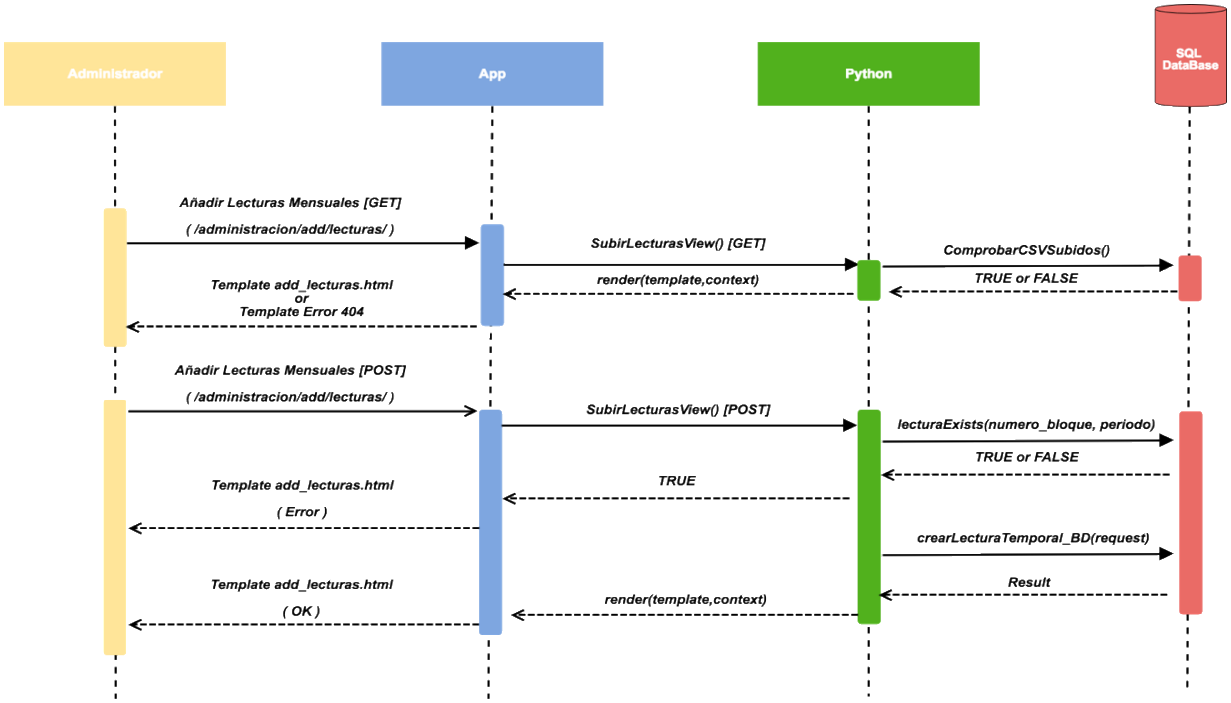
Anexo 1: Diagramas de interacción

Añadir facturas



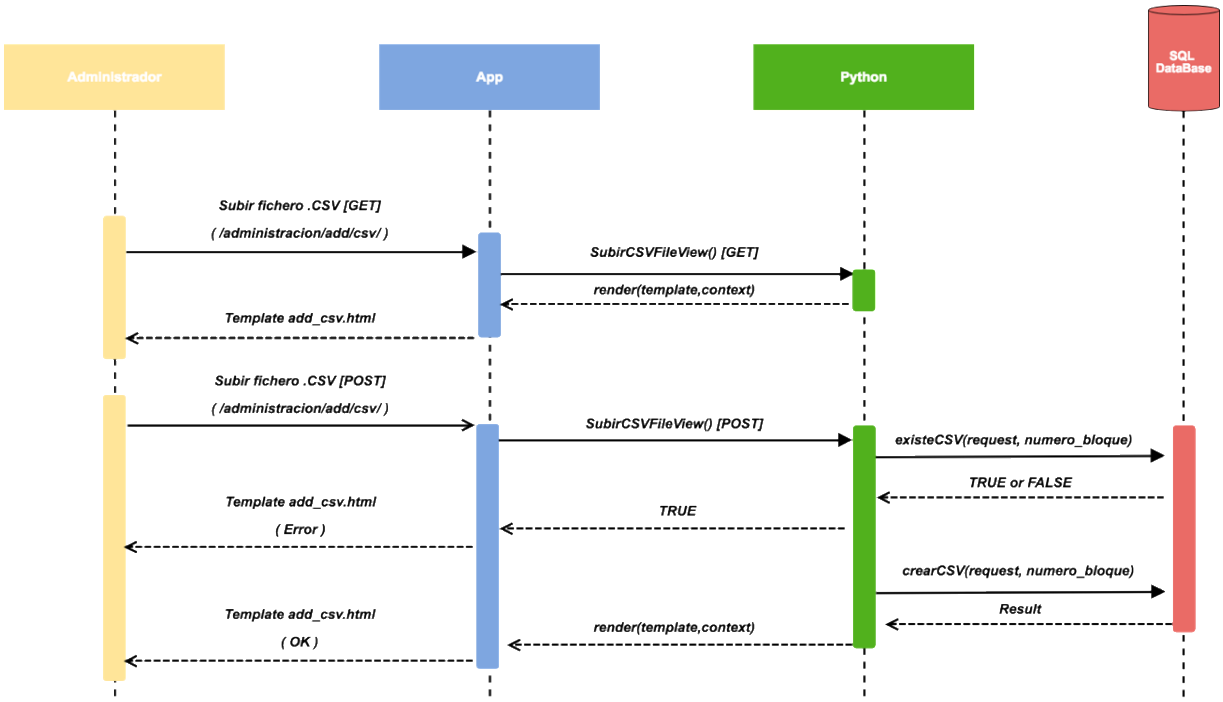
[Diagrama de interacción del requerimiento funcional 'Añadir Facturas']

Añadir lecturas



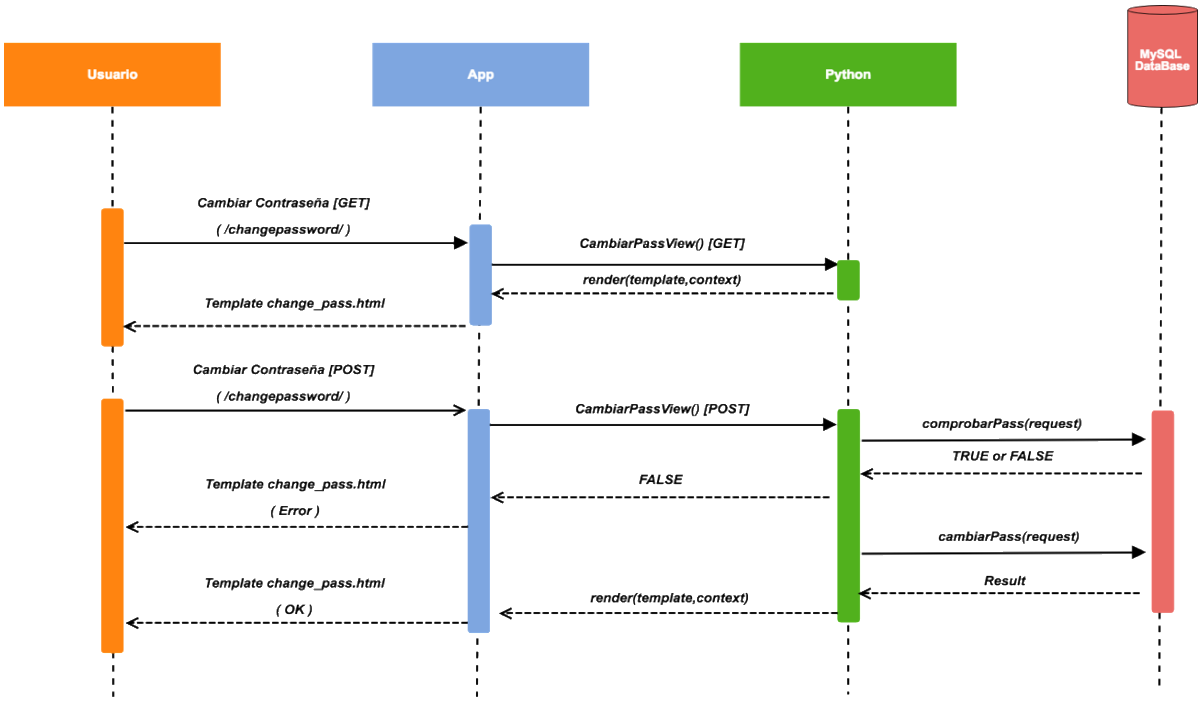
[Diagrama de interacción del requerimiento funcional 'Añadir Lecturas']

Añadir fichero .csv



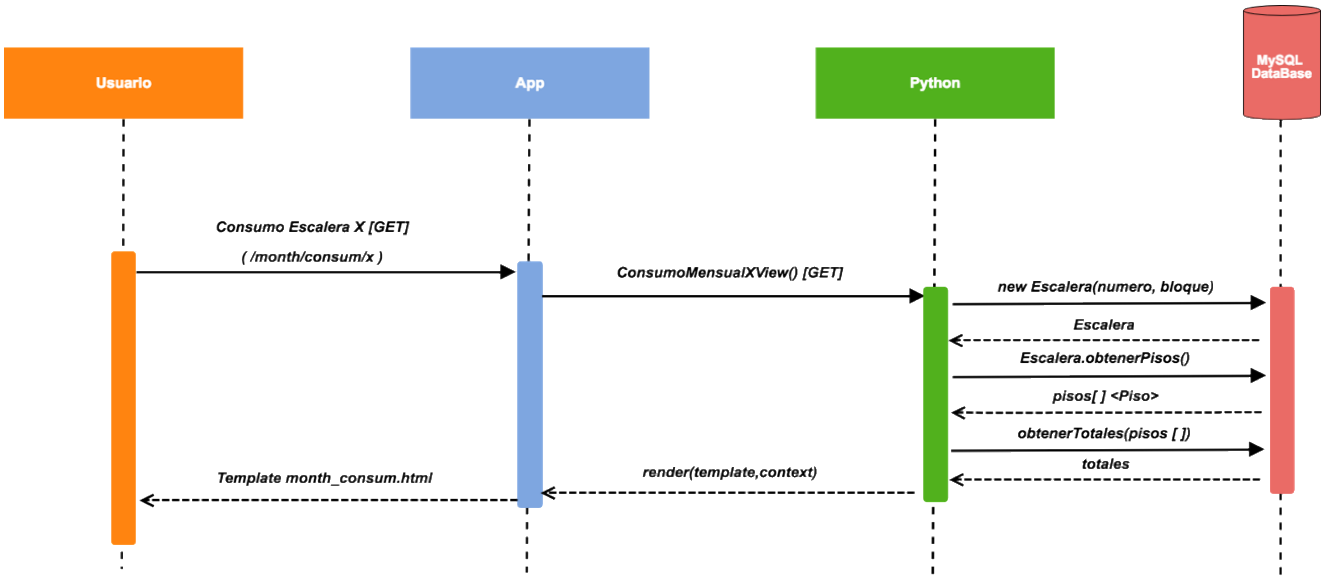
[Diagrama de interacción del requerimiento funcional 'Añadir Fichero .CSV']

Gestionar cuenta del usuario



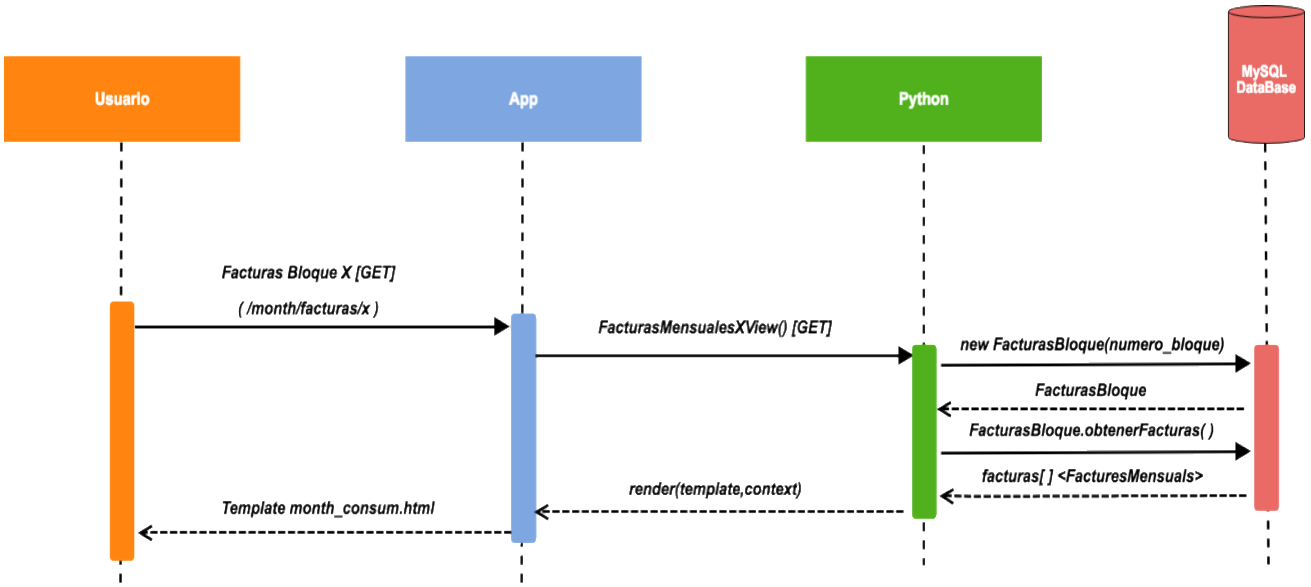
[Diagrama de interacción del requerimiento funcional 'Gestionar Cuenta Usuario']

Consultar consumos mensuales



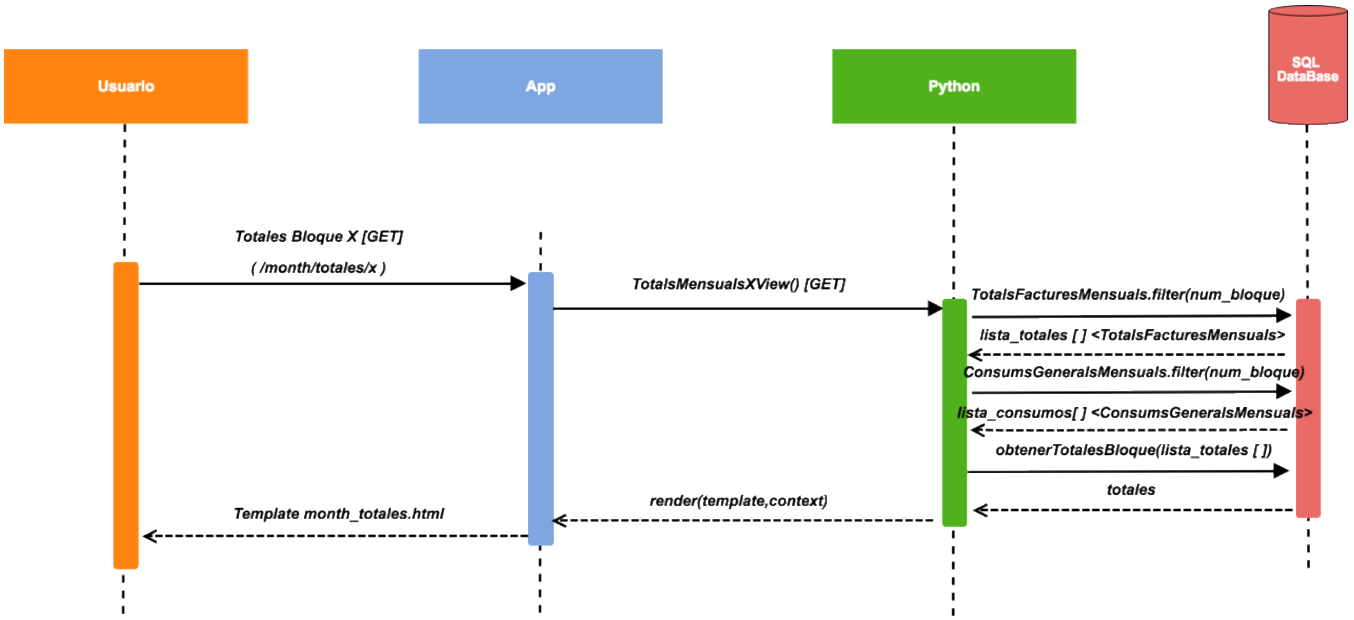
[Diagrama de interacción del requerimiento funcional 'Consultar Consumos Mensuales]

Consultar facturas mensuales



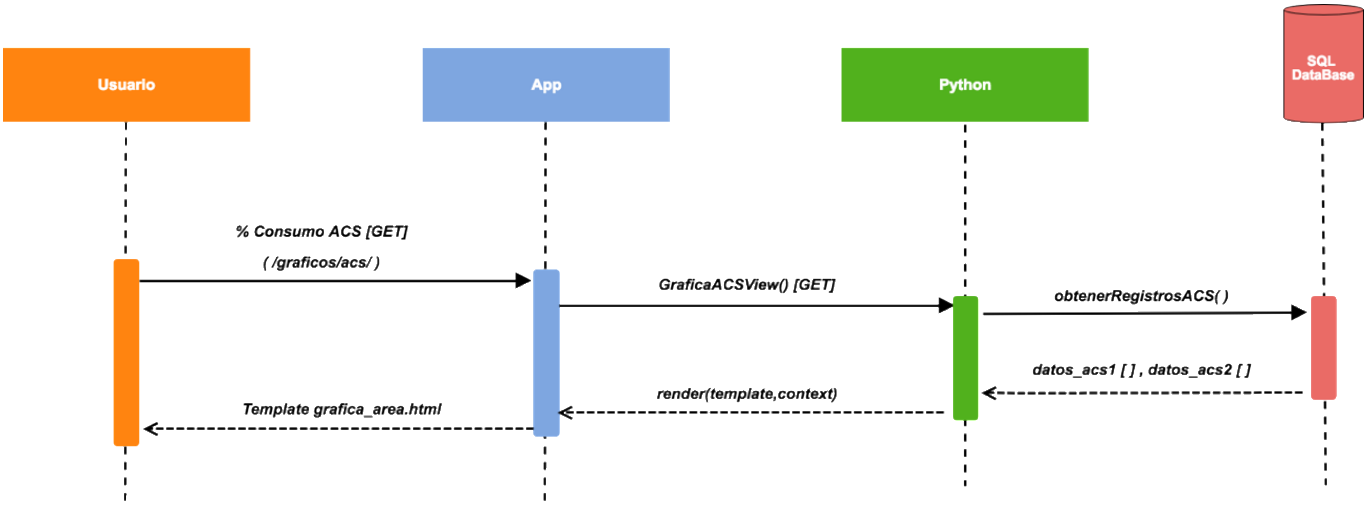
[Diagrama de interacción del requerimiento funcional 'Consultar Facturas Mensuales']

Consultar totales mensuales



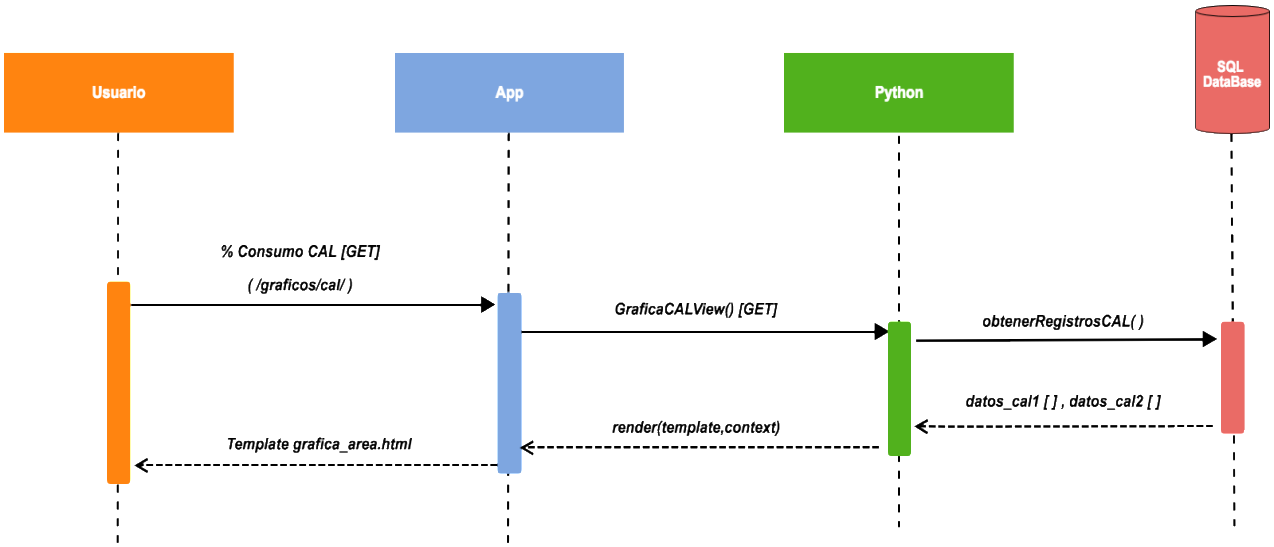
[Diagrama de interacción del requerimiento funcional 'Consultar Totales Mensuales']

Consultar Gráfico % Consumo ACS Bloque 1 vs Bloque 2



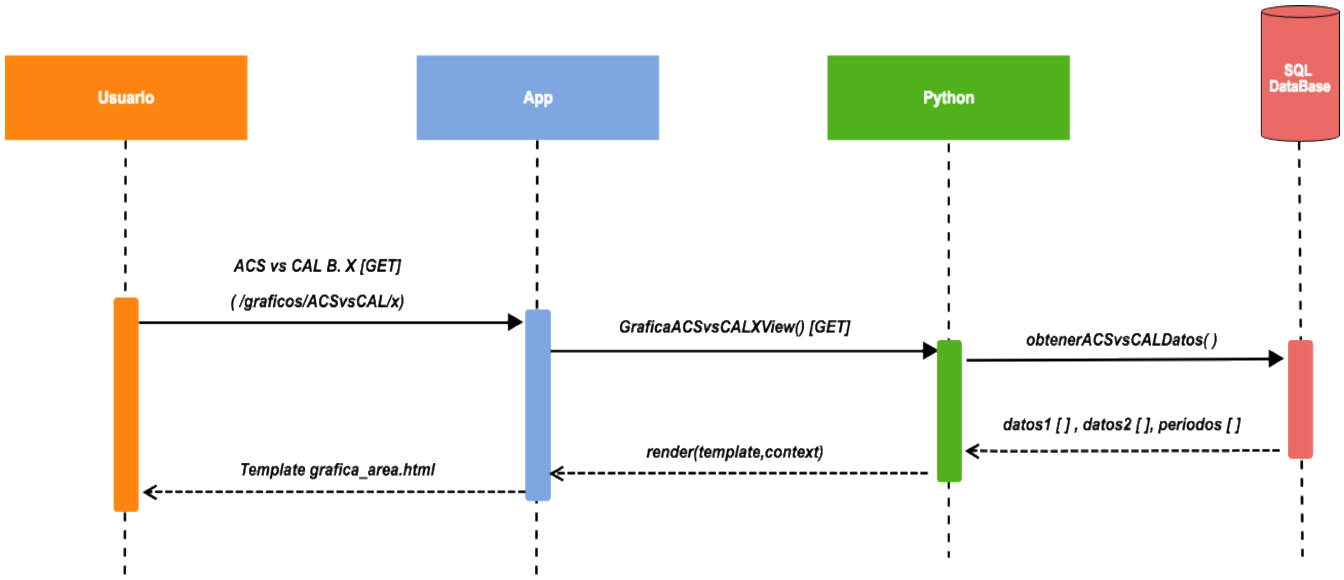
[Diagrama de interacción del requerimiento funcional 'Consumo ACS B1 vs B2']

Consultar Gráfico % Consumo CAL Bloque 1 vs Bloque 2



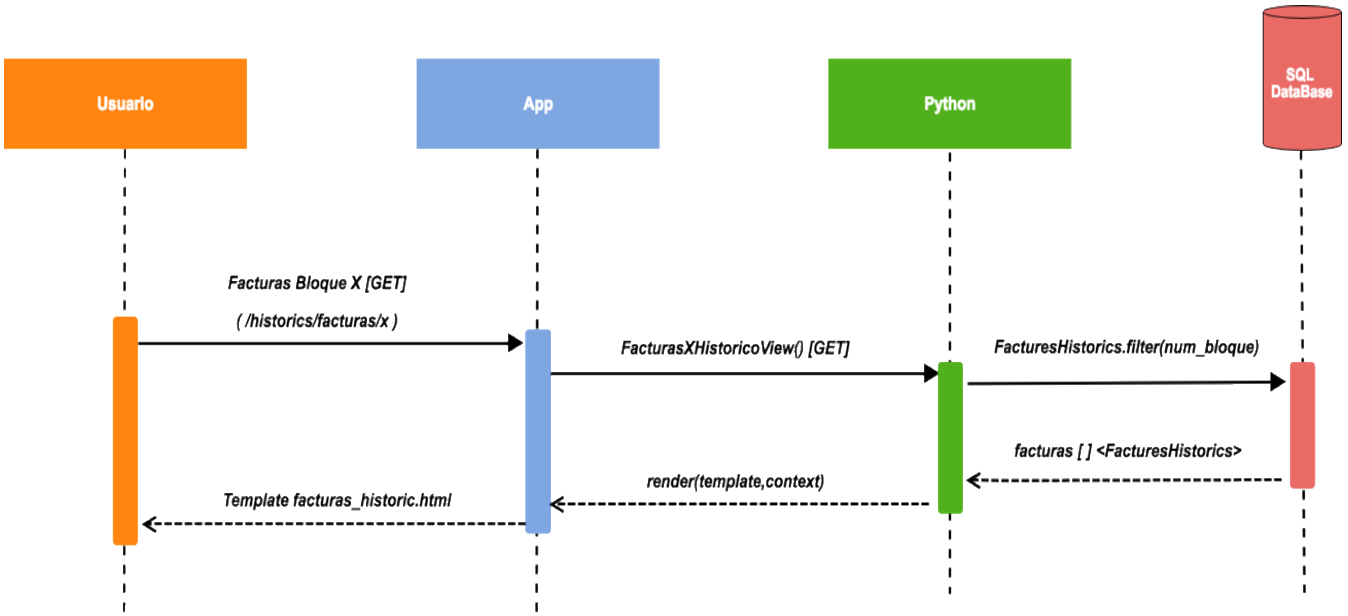
[Diagrama de interacción del requerimiento funcional 'Consumo CAL B1 vs B2']

Consultar Gráfica % Cons. ACS vs Cons. CAL (por bloque)



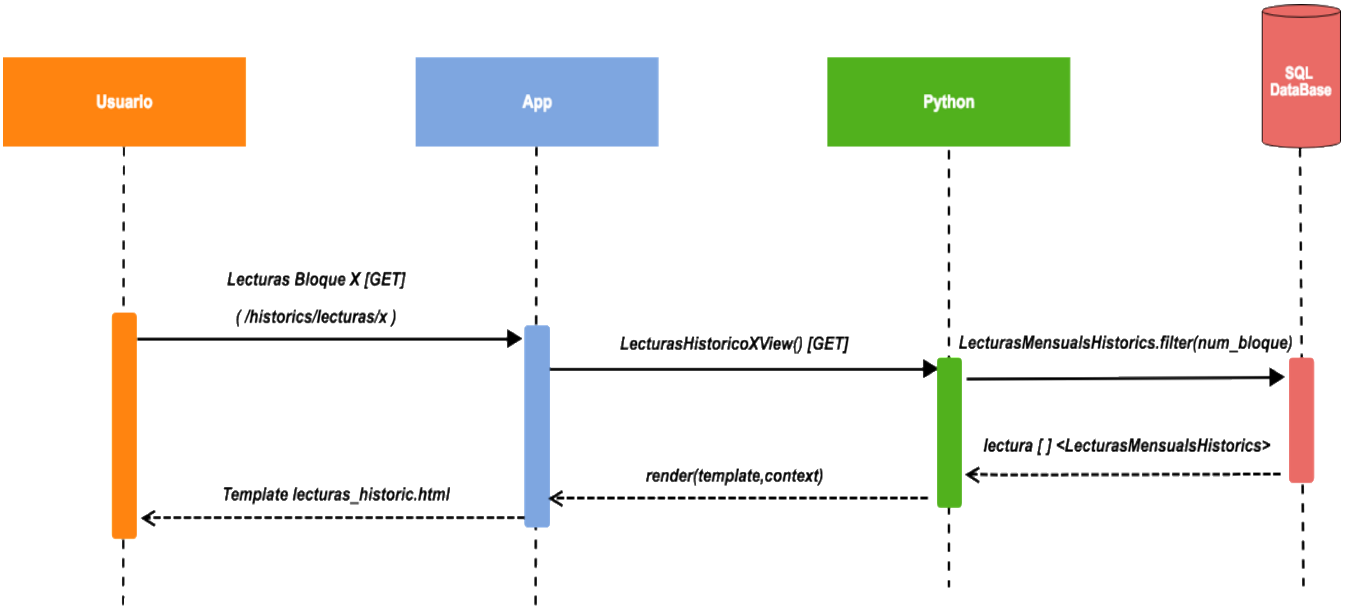
[Diagrama de interacción del requerimiento funcional 'Consumo ACS vs Consumo CAL']

Consultar histórico de facturas



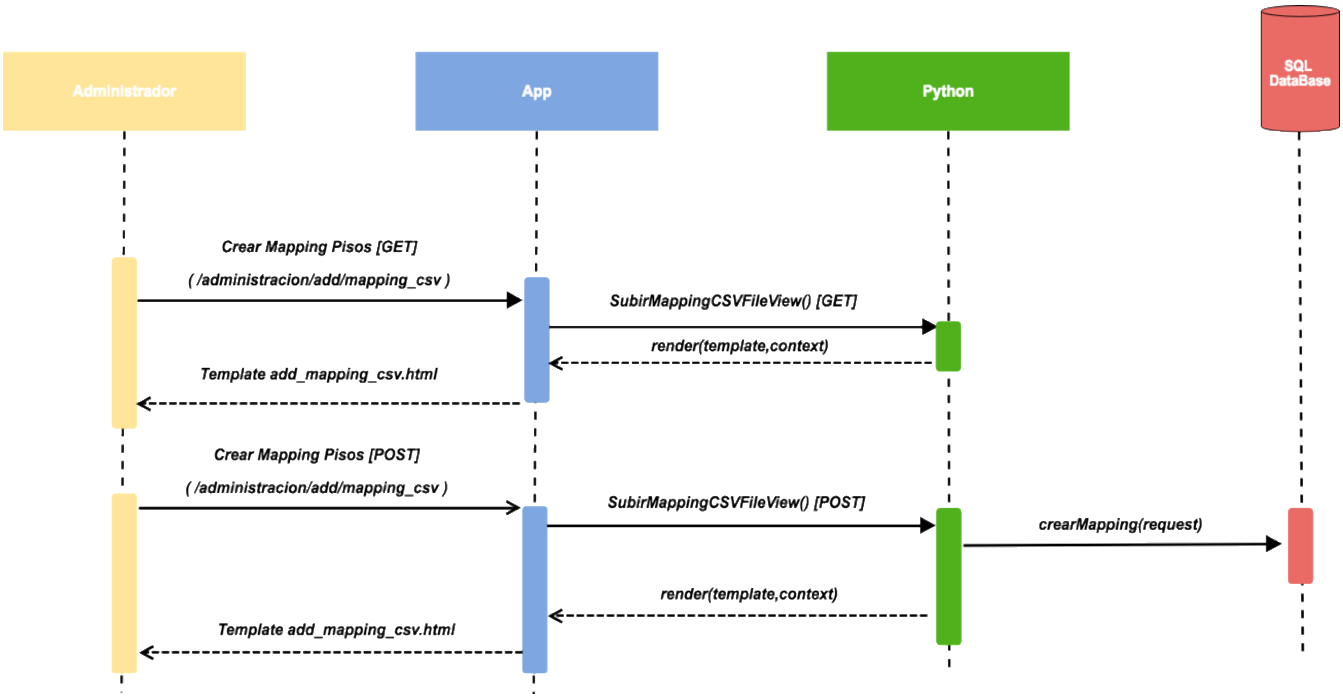
[Diagrama de interacción del requerimiento funcional 'Consultar Histórico Facturas']

Consultar histórico lecturas



[Diagrama de interacción del requerimiento funcional 'Consultar Histórico Lecturas']

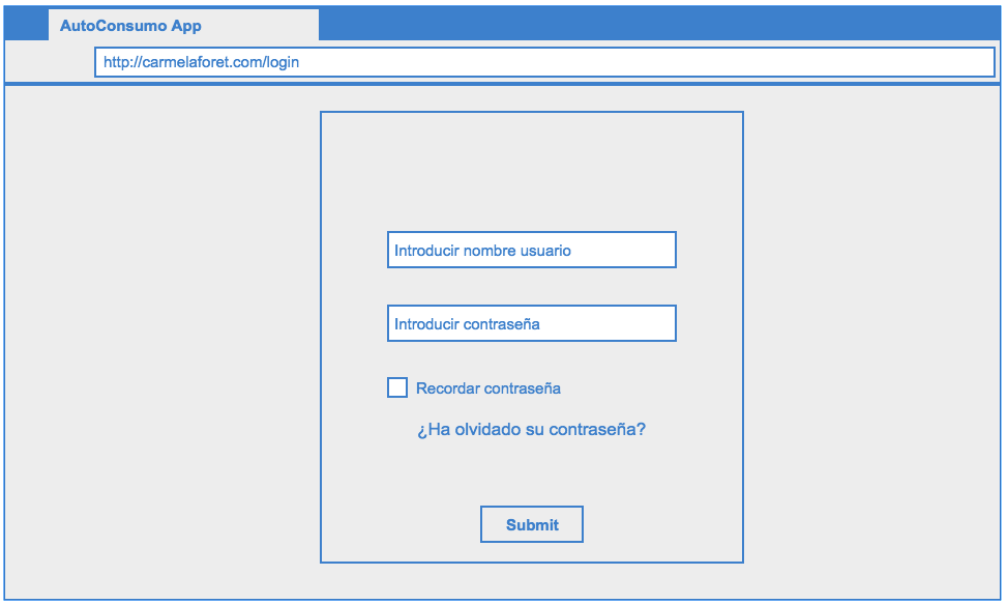
Añadir fichero .csv de mapeo



[Diagrama de interacción del requerimiento funcional 'Añadir Fichero .CSV']

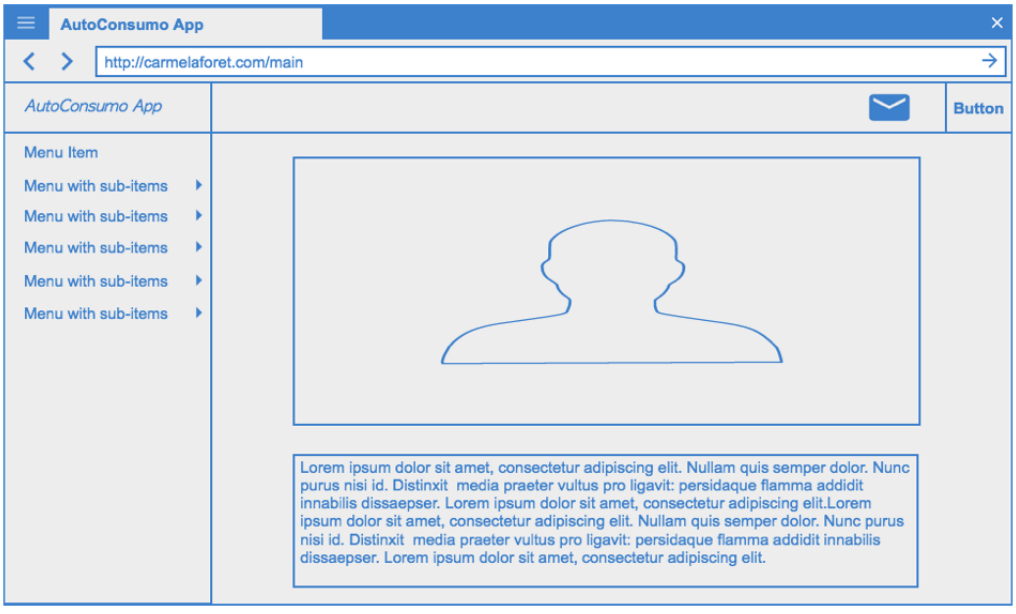
Anexo 2: Mockup's Interfaz

Mockup 'Inicio de Sesión'



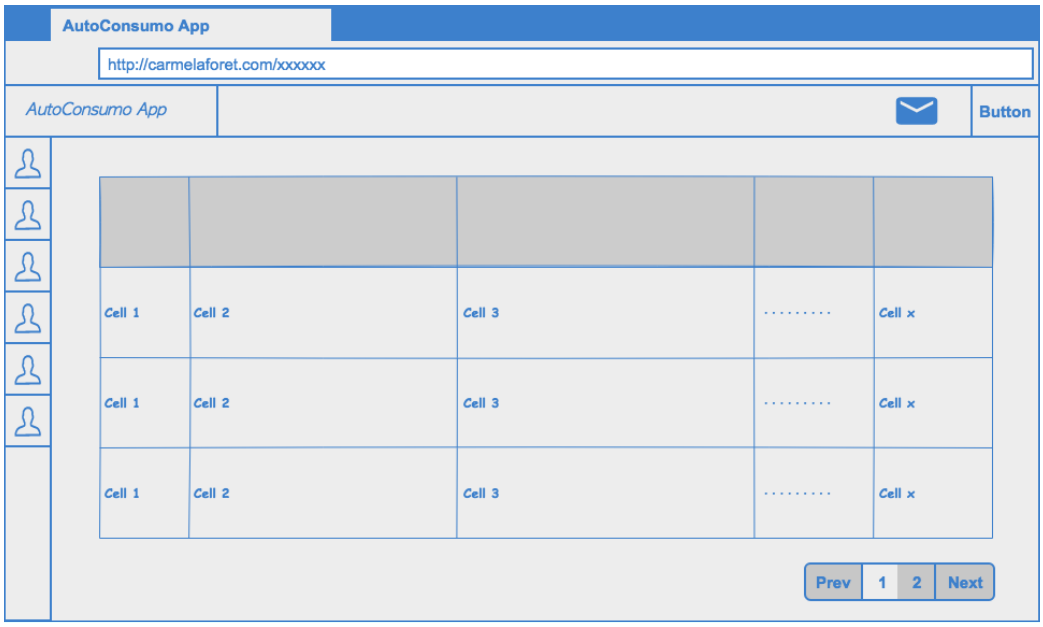
[Mockup de la pantalla destinada al 'Inicio de Sesión']

Mockup 'Menú Principal'



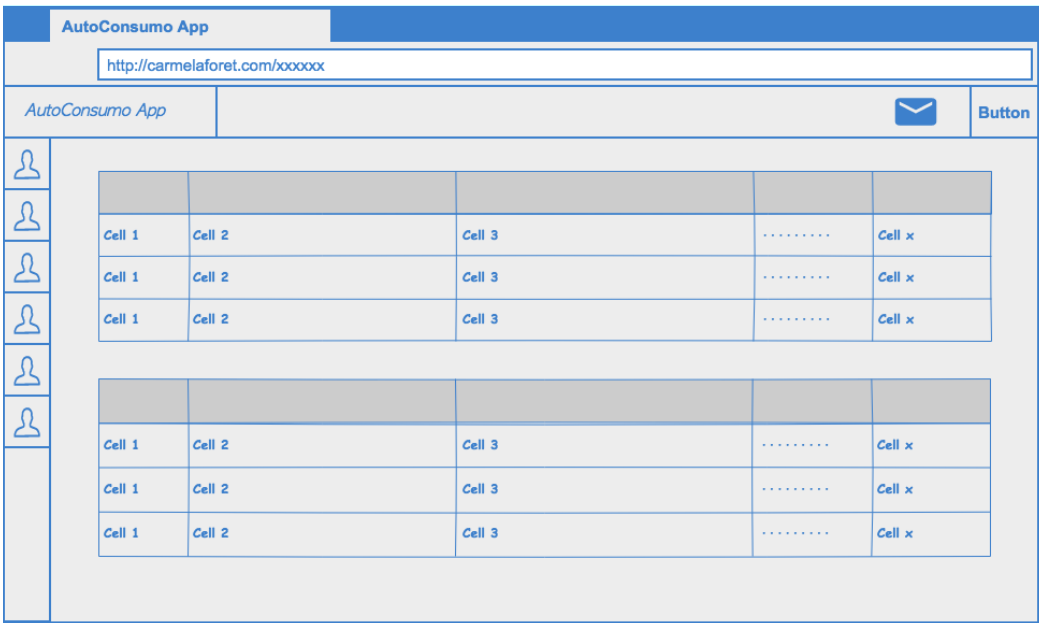
[Mockup de la pantalla destinada al 'Menú Principal']

Mockup 'Consulta Datos Única Tabla'



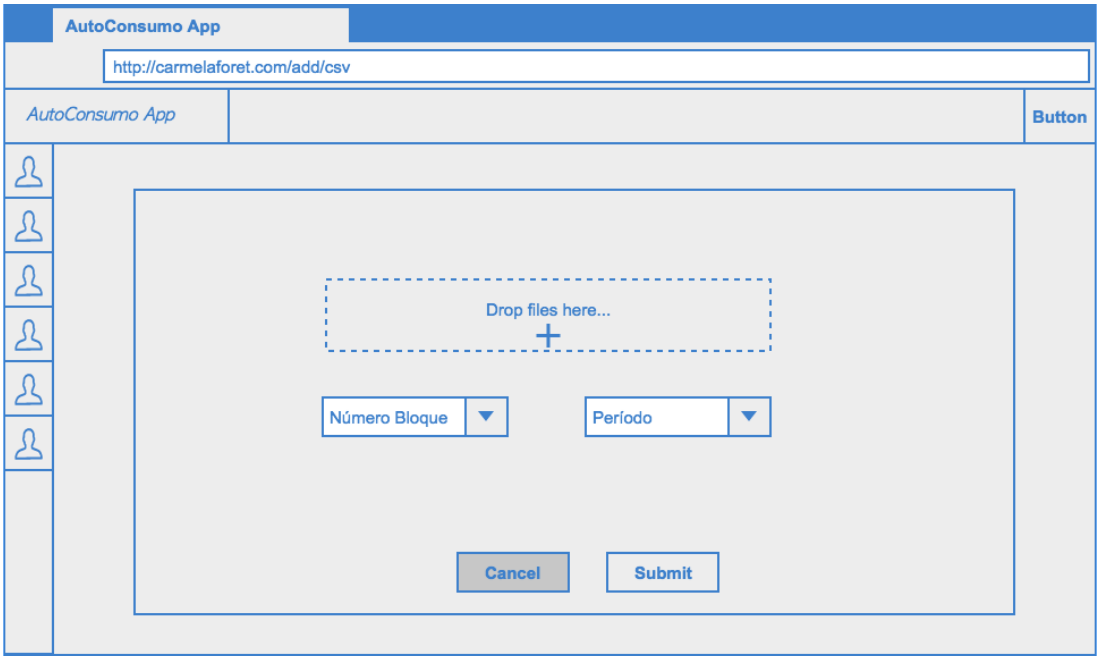
[Mockup de la pantalla destinada a la 'Consulta Datos Única Tabla']

Mockup 'Consulta Datos Dos Tablas'



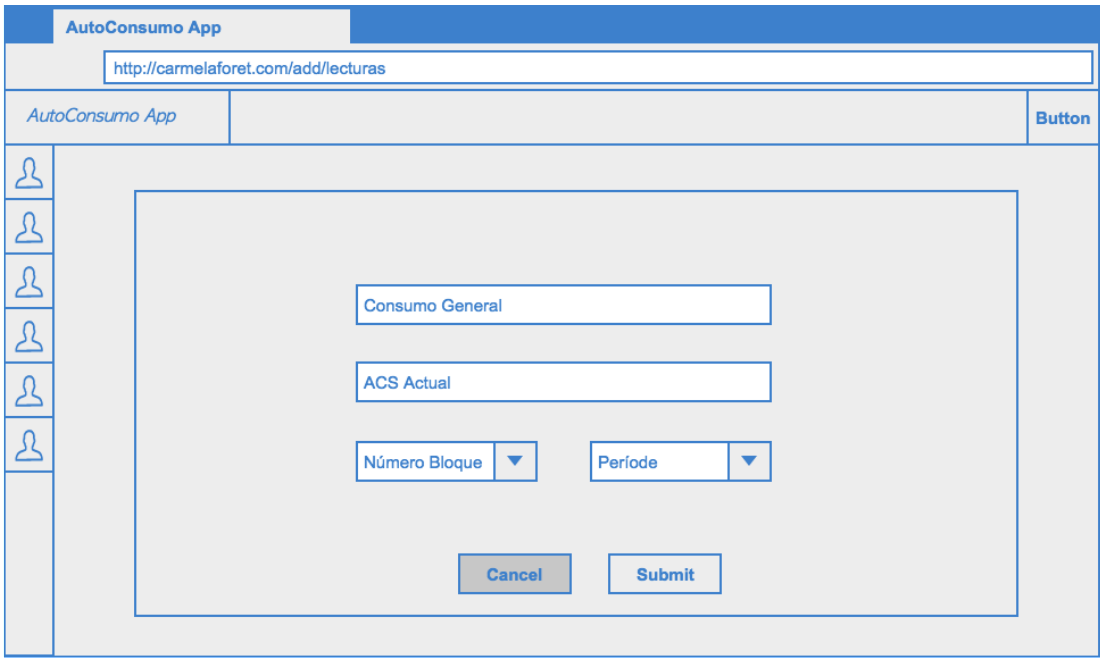
[Mockup de la pantalla destinada a la 'Consulta Datos Dos Tablas']

Mockup 'Añadir Fichero .CSV Estadístico'



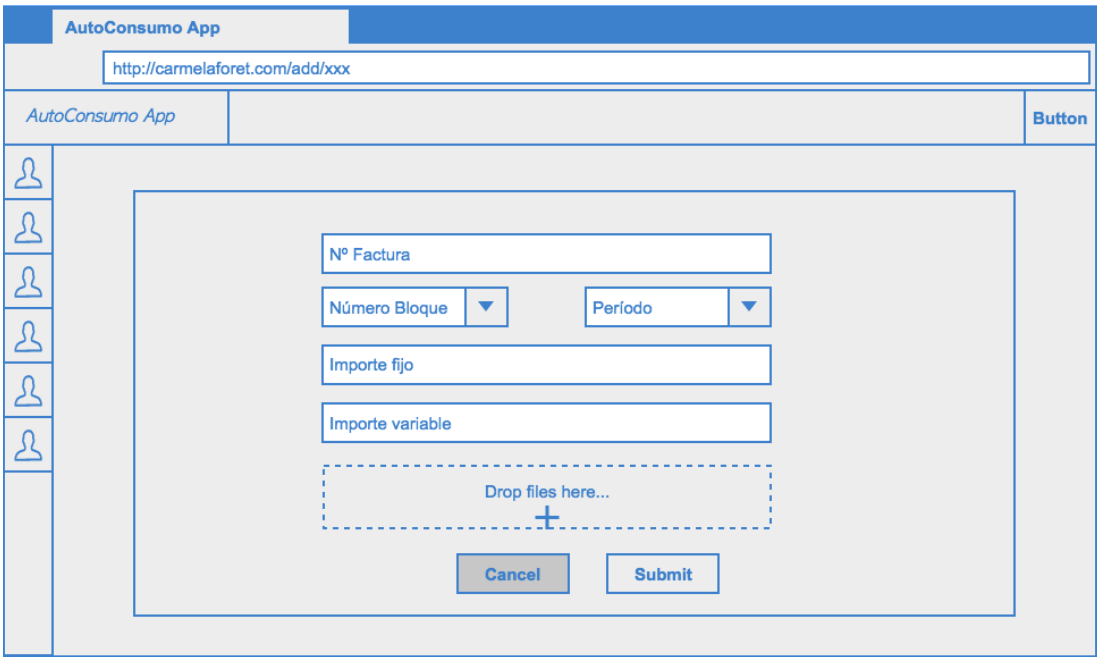
[Mockup de la pantalla destinada al 'Añadir Fichero .CSV Estadístico']

Mockup 'Añadir Lecturas Mensuales'



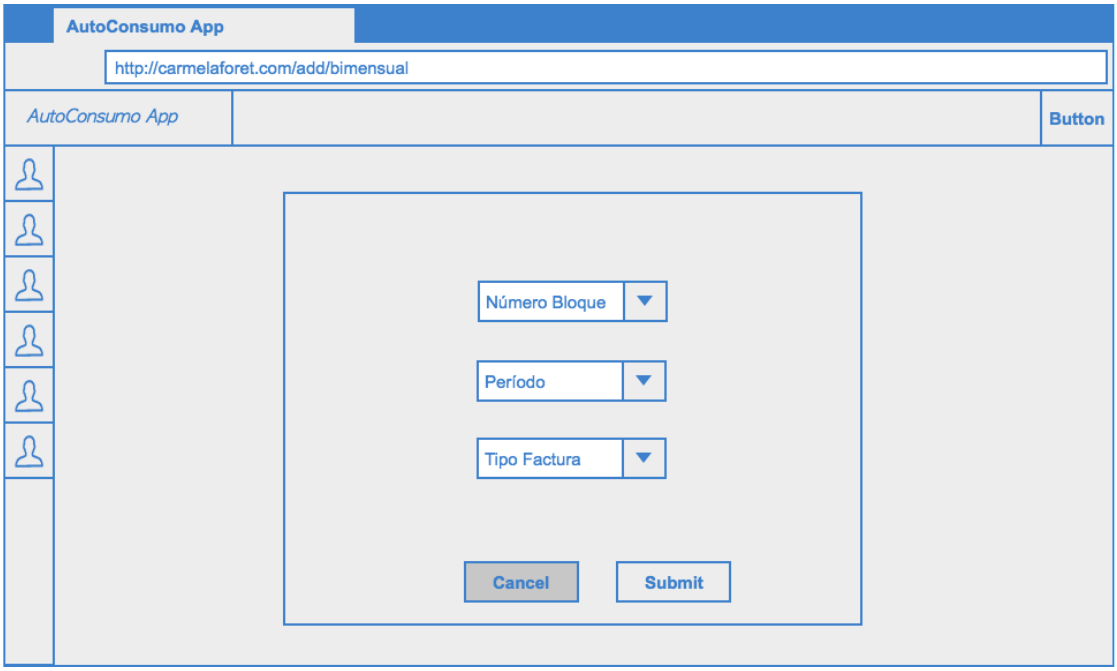
[Mockup de la pantalla destinada al 'Añadir Lecturas Mensuales']

Mockup 'Añadir Facturas Mensuales'



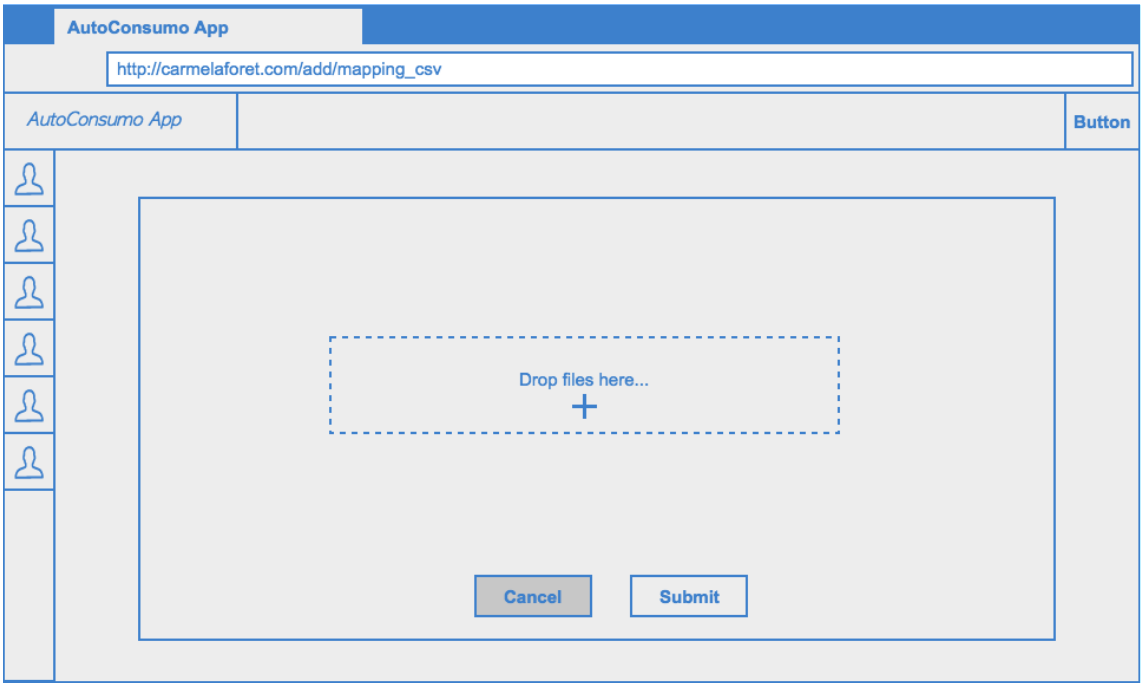
[Mockup de la pantalla destinada al 'Añadir Facturas Mensuales']

Mockup 'Añadir Facturas BiMensuales'



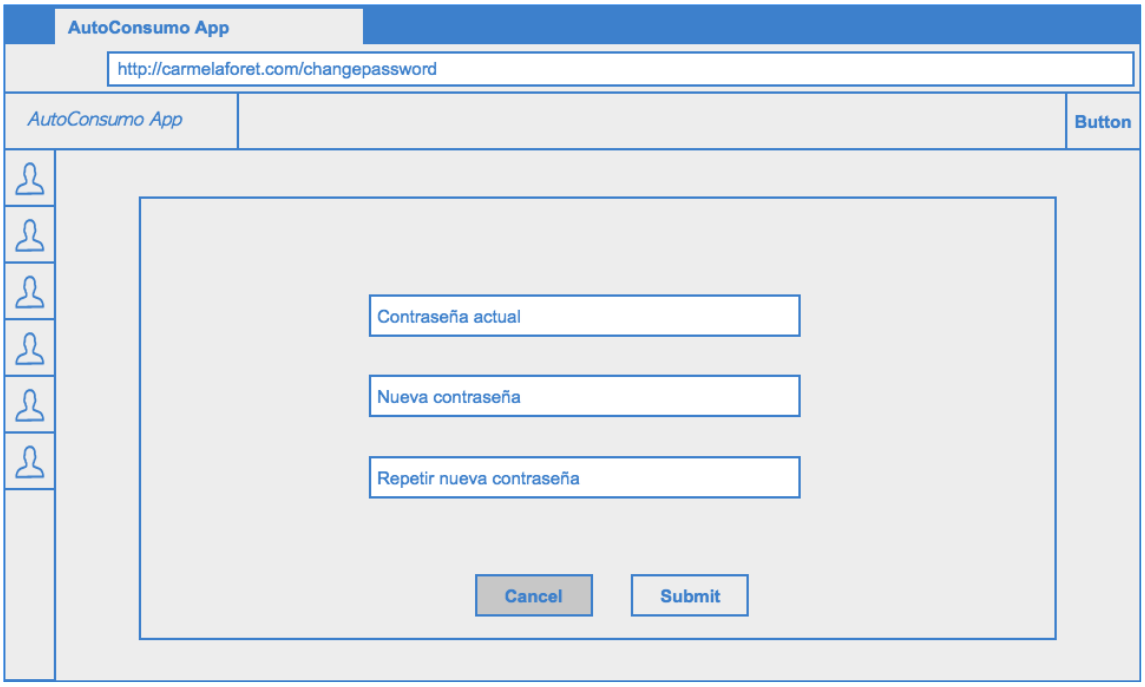
[Mockup de la pantalla destinada al 'Añadir Facturas BiMensuales']

Mockup 'Añadir Fichero .CSV Mapping'



[Mockup de la pantalla destinada al 'Añadir Fichero .CSV Mapping']

Mockup 'Gestión Contraseña Usuario'



[Mockup de la pantalla destinada al 'Gestión Contraseña Usuario']